VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Android aplikace pro směrování vozidel

Android Application for Vehicle Routing

2013 Bc. Tomáš Režnar

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Režnar**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Android aplikace pro směrování vozidel
Android Application for Vehicle Routing

Zásady pro vypracování:

V rámci této diplomové práce se bude student zabývat problematikou vybranými částmi frameworku připravovaného na VŠB-TU Ostrava pro vyhledávání nejvhodnějších trasy pro navigaci (routování) dopravního vozidla. Student se bude v rámci své práce zabývat problematikou bodu zájmů a jejich uložení a také implementací mobilního klienta pro operační systém Android.

Jednotlivé body práce jsou:
1. Prostudovat problematiku ukládání a vyhledávání bodu zájmu v směrovacích aplikacích.
2. Implementace vlastní datové struktury pro ukládání bodu zájmů.
3. Implementace vyhledávání bodu zájmů nad zvoleným řešením.
4. Nasazení implementovaného řešení pomocí webové služby.
5. Vytvoření aplikace pro operační systém Android jejíž cílem bude testování vyvíjeného frameworku pro směrování vozidel.
6. Experimentální vyhodnocení funkčnosti systémů a rychlostí vyhledávání bodu zájmu.

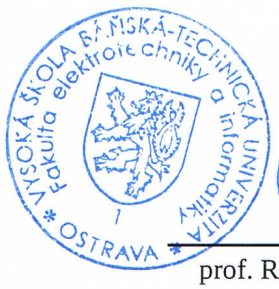
Seznam doporučené odborné literatury:

[1] Android developers, http://developer.android.com/index.html

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2013 .....

Děkuji vedoucímu mé diplomové práce Ing. Janu Martinovičovi, Ph.D. za trpělivost, ochotu a cenné rady, které pomohly tuto práci vytvořit.

Abstrakt

Tato diplomová práce se zabývá problematikou ukládání a vyhledávání bodů zájmu ve směrovacích aplikacích. V rámci práce je popsána implementace datové struktury trie, její využití pro implementaci fulltextového vyhledávání bodů zájmu a následné optimalizace vyhledávání. Dále je popsána implementace datové struktury k-d stromu a její využití pro implementaci prostorového vyhledávání bodů zájmu. Je popsáno vytvoření webové služby pro nasazení implementovaných algoritmů a využití vybraných částí připravovaného frameworku na VŠB-TU Ostrava. Je vytvořena aplikace pro operační systém Android, která testuje a demonstruje možnosti připravovaného frameworku.

Klíčová slova: fulltextové vyhledávání, prostorové vyhledávání, bod zájmu, webová služba, Android aplikace

Abstract

This thesis deals with the storage and retrieval of points of interest in routing applications. The thesis describes the implementation of the trie data structure, its use for the implementation of full-text search for points of interest and subsequent optimization search. It also describes the implementation of the k-d tree data structure and its use for the implementation of spatial search points of interest. It describes the creation of web service for deployment of implemented algorithms and the use of selected parts of the forthcoming framework for VŠB-TU Ostrava. An application for the Android operating system that tests and demonstrates the possibilities of the prepared framework was created.

Keywords: full-text search, spatial search, point of interest, web service, Android application

Seznam použitých zkratek a symbolů

2D	–	Dvojrozměrný
3D	–	Třírozměrný
API	–	Application Programming Interface
ASP.NET	–	Součást .NET Frameworku pro tvorbu webových aplikací a služeb
CSV	–	Comma-separated values
GPS	–	Global Positioning System
HTTP	–	Hypertext Transfer Protocol
JDK	–	Java Development Kit
JSON	–	JavaScript Object Notation
OS	–	Operační systém
OSM	–	OpenStreetMap
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
XML	–	Extensible Markup Language
.NET	–	Microsoft .NET Framework

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

V různých aplikacích může být potřeba vyhledávat body zájmu. Tyto aplikace mohou být různého druhu a běžet na různých platformách. Příkladem takové aplikace mohou být mapy na portálech mapy.cz, maps.google.cz a openstreetmap.org. Tyto portály umožňují fulltextově vyhledávat body zájmu zadáním textu do vyhledávacího formuláře, také umožňují zjistit adresu v daném místě mapy nebo vyhledat body zájmu v okolí.

Dalším příkladem aplikací jsou navigace, které jsou buď specializované zařízení, nebo aplikace pro mobilní zařízení. V navigacích bývá, jako jedna z funkcí možnost vyhledat různé body zájmu v okolí, například nejbližší benzínovou pumpu.

V této diplomové práci jsou popsány datové struktury trie a k-d strom a jejich využití pro fulltextové a prostorové vyhledávání bodů zájmu. Je vytvořena webová služba, která využívá tyto struktury pro vyhledávání bodů zájmu. Vytvořená webová služba je následně využita aplikací pro operační systém Android, jejíž vývoj je také součástí této práce.

1.1 Struktura práce

Práce se v sekci 2 zabývá implementací knihovny pro vyhledávání bodů zájmu. Nejprve se zabývá datovou strukturou trie a její konstrukcí. Poté jsou popsány algoritmy pro přesné a nepřesné vyhledávání v trii. Je popsán formát ukládání trie do souboru a její načítání ze souboru.

V sekci 2.2 je popsán algoritmus pro fulltextové vyhledávání bodů zájmu s využitím algoritmu pro nepřesné vyhledávání v trii. Jsou popsány vlastnosti algoritmu, chyby v uživatelem zadaném vyhledávaném textu, které se snaží algoritmus ošetřit. Dále se práce zabývá optimalizací fulltextového vyhledávání využitím různých datových struktur a úpravou způsobu procházení trie.

V sekci 2.4 je popsána datová struktura k-d strom, její konstrukce a vlastnosti. Je popsán algoritmus vyhledávání bodů zájmu v dané oblasti a vyhledání bodů zájmu nejbližších k daným souřadnicím s využitím k-d stromu. Dále je popsán formát ukládání k-d stromu do souboru a načtení k-d stromu ze souboru. Nakonec jsou provedeny testy rychlosti vyhledávání bodů zájmu v k-d stromu.

V sekci 3 se práce zabývá nasazením implementovaných algoritmů vyhledávání bodů zájmu pomocí webové služby. Webová služba také využívá připravovaný framework na VŠB-TU Ostrava pro směrování vozidel a výpočet dostupnosti.

V sekci 4 je popsán vývoj aplikace pro operační systém Android, která bude využívat implementovanou webovou službu. Jsou popsány základní vlastnosti operačního systému Android a volba verze API pro vyvíjenou aplikaci. Je popsáno zpracování dat v aplikaci a jejich kešování. Dále je popsáno uživatelské rozhraní aplikace a nakonec je experimentálně otestováno vyhledávání.

2 Ukládání a vyhledávání bodů zájmu v směrovacích aplikacích

Při vývoji směrovacích aplikací bývá potřeba navrhnout a implementovat algoritmy umožňující různé vyhledávání bodů zájmu. Bod zájmu [24] je lokace ve světě, dána svými zeměpisnými souřadnicemi, o jejíž vyhledání může mít uživatel zájem. V reálném světě může bod zájmu zaujímat velkou plochu (například stadión, jezero) nebo může být bodový (například strom).

Ve směrovacích aplikacích je většinou potřeba uživatele nasměrovat na konkrétní lokaci (například v případě stadiónu k vstupní bráně), proto se body zájmu, které zaujímají větší rozlohu, uloží do několika samostatných bodů (například v případě stadiónu budou jednotlivými body vstupní brány, které jsou pro vyhledání cesty důležitými body).

Často požadovanou funkčností bývá fulltextové vyhledávání bodů zájmu. Při tomto způsobu vyhledávání uživatel zadá text, na jehož základě mu budou nabídnuty body zájmu seřazené podle podobnosti názvu jednotlivých bodů se zadaným textem. Takové vyhledávání se uplatní v případech, kdy uživatel zná alespoň část názvu hledaného bodu, ale neví, kde se přesně nachází nebo nechce daný bod na mapě hledat. Příkladem případu užití je vyhledávací řádek na Google mapách a mnoha jiných.

Další častou funkcí bývá vyhledání nejbližších bodů zájmu (vyhledávání nejbližších sousedů [3]). Při tomto vyhledávání zadá uživatel střed vyhledávání (například jeho aktuální polohu zjištěnou pomocí GPS) a typ bodů zájmu, které chce nalézt. V praxi se tento typ vyhledávání uplatní například v případě, kdy uživatel potřebuje zjistit, na jaké se nachází adrese, potřebuje zjistit nejbližší bankomat, restauraci a podobně.

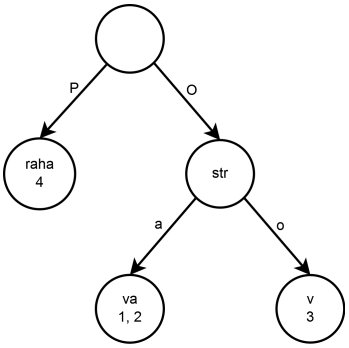
Poslední důležitou funkcí je vyhledání bodů zájmu v konkrétní oblasti. Při tomto vyhledávání je zadána oblast hledání a typ bodů zájmu. Uplatní se například v aplikacích, kde se na mapový podklad vykreslují uživateli body zájmu, o které se uživatel zajímá (například restaurace v okolí).

2.1 Datová struktura trie

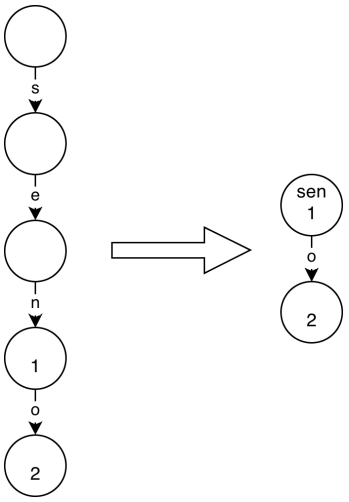
Trie [12] se používá pro uchovávání množiny asociací. V případě použití pro vyhledávání bodů zájmu je klíčem slovo a hodnotou je množina všech bodů zájmu, které obsahují dané slovo ve svém názvu. Trie umožňuje rychlé vyhledávání textu včetně vyhledávání variací hledaného slova, proto může být použita jako základ fulltextového vyhledávání.

Je to nelineární datová struktura - strom. Každý uzel trie odpovídá konkrétnímu klíči a pokud je s daným klíčem asociována hodnota, pak je obsažena v daném uzlu. Každý uzel má přiřazen prefix. Prefix má nenulovou délku, kromě kořene trie, jehož prefix je prázdný řetězec.

Každý list musí obsahovat hodnotu asociovanou s jeho klíčem. Klíč uzlu je zřetězením všech prefixů předků uzlu včetně prefixu uzlu samotného v pořadí od kořene. Každý uzel má maximálně tolik potomků, kolik je velikost abecedy použité pro vytváření klíčů. Jednotlivé děti uzlu jsou rozlišitelné na základě prvního znaku přiřazeného prefixu, proto každé dítě uzlu musí mít unikátní první znak prefixu.



Obrázek 1: Ukázka struktury trie



Obrázek 2: Ukázka trie před a po komprimaci

Na obrázku 1 je ukázka trie, do které je vložena množina asociací {Ostrava: 1, Ostrava: 2, Ostrov: 3, Praha: 4}. Pro zdůraznění unikátnosti prvního znaku prefixu uzlu, je první znak zobrazen na hraně vedoucí směrem od rodiče k uzlu a zbytek prefixu je zobrazen v samotném uzlu.

Prefix slova je úsek znaků, kterými dané slovo začíná. Délka prefixu může být minimálně nulová a maximálně rovna délce daného slova. Například slovo Ostrava má prefixy: „“ (prázdné slovo), O, Os, Ost, Ostr, Ostra, Ostrav, Ostrava.

Komprimovaná trie je struktura vycházející z trie, má oproti základní trii menší počet uzlů dosažený bezztrátovou kompresí. Komprimovaná trie splňuje následující podmínku [11]: každý uzel, který není listem nebo kořenem, má se svým klíčem asociovánu hodnotu nebo má minimálně dvě děti. Komprimací trie dojde k eliminaci uzlů, které by zbytečně zvyšovaly hloubku stromu, ale nepřidávaly by žádnou další informaci. Na obrázku 2 je zobrazena trie před a po komprimaci se vstupem množiny asociací {sen: 1, seno: 2}.

2.1.1 Konstrukce komprimované trie

Při konstrukci komprimované trie lze volit mezi dvěma přístupy. Buď lze trii konstruovat nekomprimovaně a po vložení všech prvků množiny asociací provést komprimaci, nebo lze kompresi provádět přímo při vkládání. Provedením metody ve výpisu kódu 1 pro všechny prvky asociativní množiny vznikne komprimovaná trie. Metoda provádí kompresi přímo při vkládání. Samotná metoda je definována v třídě $Node$, která představuje uzel stromu trie.

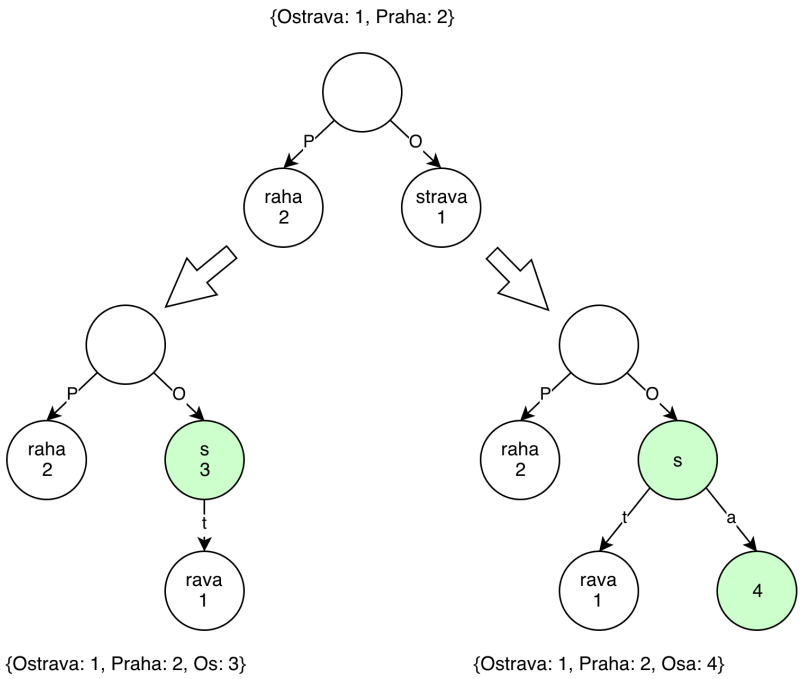
```
1  public void Add(string key, TValue value)
2  {
3     if (key.Equals(""))
4     {
5        values.Add(value);
6     }
7     else
8     {
```

Nejprve se zkontroluje, jestli je klíč nulové délky. Pokud ano, pak se hodnota vloží do uzlu. Pokud klíč není prázdný řetězec, pak je nutné určit vhodné dítě. Může nastat několik případů. Při určení vhodného dítěte je využito vlastnosti uzlu trie, kdy každé dítě uzlu musí mít unikátní počáteční znak přiřazeného prefixu.

```
9        AChildrenDictionary<TValue>.Association assoc;
10       if (children.Get(key[0], out assoc))
11       {
12          if (key.StartsWith(assoc.Prefix.Whole, StringComparison.Ordinal))
13          {
14             assoc.Node.Add(key.Substring(assoc.Prefix.Whole.Length), value);
15          }
16          else
17          {
```

Objekt *assoc* definovaný na řádku 9. obsahuje informace o přiřazení prefixu k uzlu. Pokud klíč začíná prefixem některého dítěte (řádek 12.), pak se z klíče odstraní zleva prefix dítěte a dojde k rekurzivnímu volání metody.

```
18             int i = NumberOfIdenticalCharacterFromLeft(key, assoc.Prefix.Whole);
19             TrieNode<TValue> n = assoc.Node;
20             assoc.Node = new TrieNode<TValue>();
21             assoc.Node.AddChildren(assoc.Prefix.Whole.Substring(i), n);
22             assoc.Prefix = new Prefix(assoc.Prefix.Whole.Substring(0, i));
23
24             if (key.Length == i)
25             {
26                assoc.Node.Add("", value);
27             }
28             else
29             {
30                n = new TrieNode<TValue>();
31                n.Add("", value);
32                assoc.Node.AddChildren(key.Substring(i), n);
33             }
```



Obrázek 3: Ukázka změny trie při vkládání

Určí se počet shodných znaků klíče a prefixu dítěte (řádek 18.), vytvoří se nové dítě s prefixem rovným shodné části klíče a prefixu původního dítěte. Nové dítě nahradí původní dítě, které se stane jeho dítětem (řádek 19. až 22.). Z prefixu původního dítěte se odstraní jeho shodná část s klíčem (řádek 21.).

Pokud se celý klíč shodoval s částí prefixu dítěte, pak je vkládaná hodnota vložena do nového dítěte (řádek 26.), jinak je vytvořen nový uzel s prefixem neshodné části klíče (řádek 30. až 32.). Na obrázku 3 je znázorněna změna trie při vkládání, původní trie obsahovala množinu asociací {Ostrava: 1, Praha: 2} a poté byly vloženy asociace (Os: 3) a (Osa: 4).

Nakonec se ošetří situace, kdy žádné z dětí nelze použít pro daný klíč a musí se vytvořit nové dítě s prefixem rovným klíči.

```
34          }
35        }
36        else
37        {
38          TrieNode<TValue> n = new TrieNode<TValue>();
39          n.Add("", value);
40          AddChildren(key, n);
41        }
42      }
43    }
```

Výpis 1: Kód metody pro vkládání do trie

2.1.2 Vyhledávání v komprimované trii

Pro realizaci fulltextového vyhledávání bodů zájmu s využitím trie je nutné implementovat operace přesného a nepřesného vyhledávání v trii. Přesné vyhledávání v trii vrátí uzel odpovídající zadanému klíči, využije se při vyhledávání bodů zájmu v případech, kdy uživatel zadá v textu libovolný počet čísel.

Zadaná čísla se vyhledávají přesně, protože se nepředpokládá, že by uživatel chtěl nalézt adresu s jiným číslem popisným, než zadaným (číslice může samozřejmě v bodu názvu být použita i v jiných případech, například v názvu restaurace).

Při přesném vyhledávání v trii se postupně prochází stromem tak, že se volí cesta, která odpovídá prefixu zadaného klíče. Průchod stromem končí v okamžiku, kdy se nalezne uzel, jehož klíč je shodný s vyhledávaným klíčem nebo pokud neexistuje potomek uzlu, jehož klíč je prefixem hledaného klíče.

Definice 2.1 *„Levenshteinova vzdálenost $d_L(v, w)$ mezi dvěma řetězci $v, w \in \sum^*$, je minimální počet editačních operací výměna (replace), vložení (insert) a zrušení (delete) ke konverzi v na w." [13, str. 4]*

Nepřesné vyhledávání v trii se využije při vyhledávání jednotlivých slov zadaných uživatelem, při vyhledávání bodů zájmu. Vy výpisu kódu 2 je metoda pro nepřesné vyhledávání v trii. Metoda je definována v třídě $Node$. Algoritmus metody je rekurzivně prováděný na uzlech stromu, dokud nedojde k dosažení maximální Levenshteinovy vzdálenosti. Vstupní parametry metody jsou:

1. key - hledaný klíč (slovo v textu zadaném uživatelem), klíč se postupně zkracuje při rekurzi metody.
2. $maxLevenshteinDistance$ - maximální Levenshteinova vzdálenost mezi hledaným klíčem a klíči nalezených uzlů.
3. $currentLevenshteinDistance$ - aktuální Levenshteinova vzdálenost, která se zvyšuje v průběhu rekurze.
4. $result$ - množina nalezených uzlů včetně informace o Levenshteinově vzdálenosti mezi klíčem uzlu a hledaným klíčem. Tento parametr slouží jako výstup metody.

Na řádku 3 se zjistí, jestli se aktuální uzel má přidat do množiny výsledku. Uzel se přidá, pokud nebude překročena maximální Levenshteinova vzdálenost. Pokud má uzel děti (řádek 11), tak se pro každé dítě (řádek 14) projde množina následujících klíčů (řádek 16) a pro každý následující klíč se rekurzivně zavolá nepřesné vyhledávání v uzlu trie (řádek 18).

```
1  private Result<TValue> Search(string key, byte maxLevenshteinDistance, byte
       currentLevenshteinDistance, Result<TValue> result)
2  {
3    if (key.Equals("") || key.Length + currentLevenshteinDistance <= maxLevenshteinDistance)
4    {
5      if (values != null)
```

```
6       {
7           result.Add(this, (byte)(key.Length + currentLevenshteinDistance));
8       }
9   }
10
11  if (children.Count() > 0)
12  {
13      byte localMaxLd = (byte)(maxLevenshteinDistance − currentLevenshteinDistance);
14      foreach (var assoc in children.GetAll())
15      {
16          foreach(var nextKey in BestNextKey(key, assoc.Prefix.Whole, localMaxLd))
17          {
18              assoc.Node.Search(nextKey.key, maxLevenshteinDistance, (byte)(
                        currentLevenshteinDistance + nextKey.levenshteinDistance), result);
19          }
20      }
21  }
22  return result;
23 }
```

Výpis 2: Kód metody pro nepřesné vyhledávání v trii

Množina následujících klíčů je vytvářena metodou $BestNextKey$. Vstupem metody jsou řetězce α, β a maximální Levenshteinova vzdálenost l_{max}. Výstupem metody je množina řetězců R, která splňuje formuli 1. Množina R tedy obsahuje takové sufixy klíče α, pro které je Levenshteinova vzdálenost mezi prefixem uzlu β a zbytkem klíče, po odstranění sufixu, minimální.

$$\begin{aligned} &\forall\gamma_1 \in R\, \forall\gamma_2 \in R\, \forall\gamma_3 \notin R \rightarrow \delta_1\gamma_1 = \alpha \ \wedge\ \delta_2\gamma_2 = \alpha \ \wedge\ \delta_3\gamma_3 = \alpha \wedge \\ &\wedge\, d_L(\delta_1, \alpha) = d_L(\delta_2, \alpha) \ \wedge\ d_L(\delta_1, \alpha) < d_L(\delta_3, \alpha) \ \wedge\ d_L(\delta_1, \alpha) \leq l_{max} \end{aligned} \tag{1}$$

2.1.3 Formát ukládání trie

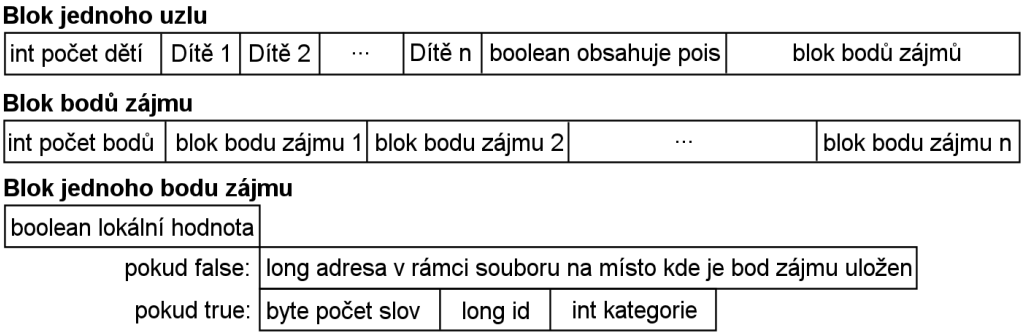
Aby se jednou vytvořená trie nemusela vytvářet znovu, například při restartu webové služby, lze ji uložit na disk do souboru a poté ze souboru načíst do paměti tak, aby rychlost načtení byla vyšší než rychlost vygenerování celé nové trie.

Při načítání ze souboru se nemusí kontrolovat podmínky pro splnění komprimace trie, je předem znám počet dětí uzlu a počet hodnot v uzlu, které lze využít pro vytvoření seznamů o přesné velikosti (oproti dynamickému zvětšování seznamu při vytváření nové trie).

Na obrázku 4 je zobrazen formát ukládání trie na disk. Základní částí je blok jednoho uzlu. Uložením kořenového uzlu trie dojde k uložení celého stromu. Blok uzlu obsahuje informaci o počtu dětí následované bloky jednotlivých dětí (v případě, že uzel děti nemá, žádné bloky dětí nenásledují).

Každý blok dítěte je ve formátu bloku jednoho uzlu. Za bloky dětí následuje příznak, informující jestli daný uzel obsahuje nějaké body zájmu. Pokud je příznak true, pak následuje blok bodů zájmu.

Blok bodů zájmu obsahuje informaci o počtu bodů zájmu, následovanou bloky jednotlivých bloků bodů zájmu. Každý blok bodu zájmu obsahuje příznak, zdali je bod



Obrázek 4: Formát uložení trie na disku

zájmu uložen lokálně, nebo v jiném místě souboru. Pokud je příznak false, pak následuje adresa místa v souboru, ve kterém je daný bod zájmu uložen. Pokud je příznak true, pak následují data bodu zájmu.

Příznak, informující jestli je bod zájmu uložen lokálně nebo v jiném místě souboru, zajišťuje, aby jeden bod zájmu nebyl uložen opakovaně na více místech (například pokud bude do trie vkládán bod zájmu se jménem „Ostrava Poruba“, pak se bude v trii nacházet v uzlu s klíčem „Ostrava“ a v uzlu s klíčem „Poruba“). Z příznaku lze snadno zjistit, že daný bod zájmu byl již dříve načtený a použije se jeho existující instance.

2.2 Fulltextové vyhledání bodů zájmu

Fulltextové vyhledání bodů zájmu umožňuje vyhledat body zájmu na základě zadání vstupního textu, který nemusí být shodný se skutečnými jmény bodů zájmu a může obsahovat různé chyby. Při vyhledávání bodů zájmu může mít uživatel k dispozici různé typy klávesnic a znakových sad, proto se při vyhledávání bude ignorovat diakritika. Seznam chyb v uživatelem zadaném vyhledávaném řetězci, které by měl algoritmus umět ošetřit:

1. Text zadaný s diakritikou i bez diakritiky.
2. Text zadaný libovolně malými nebo velkými písmeny.
3. Vynechání písmen ve slově, v libovolném místě (na počátku, uprostřed, na konci).
4. Nadbytečné písmena ve slově (vzniklé například nechtěným stisknutím klávesy).
5. Překlep (vloženo jiné písmeno, než správné).

Vyhledané body zájmu jsou seřazeny na základě podobnosti jejich názvů s zadaným textem, které by mělo probíhat na základě:

1. Levenshteinovy vzdálenosti mezi zadanými slovy textu a slovy názvů bodů zájmu.
2. Počtu zadaných slov, které jsou obsaženy v názvu konkrétního bodu zájmu (například uživatel zadá „Moravská Ostrava“, pak shoda s bodem zájmu „Ostrava“ je pouze jedno slovo).
3. Počtu bodů zájmu, které obsahují dané slovo (například slovo „Republika“ má nižší váhu než slovo „Ostrava“).
4. Počtu zadaných slov a počtu slov v názvech bodů zájmu (například při vyhledávání textu „Ostrava“ je ve výsledku bod zájmu „Ostrava“ výše než bod zájmu „Moravská Ostrava“).

2.2.1 Algoritmus fulltextového vyhledávání bodů zájmu s využitím trie

Fulltextového vyhledávání bodů zájmu bude využívat přesné a nepřesné vyhledávání v trii, popsané v kapitole 2.1.2. Fulltextové vyhledávání je popsáno v algoritmu 2.1. Algoritmus počítá pro každý nalezený bod zájmu rank, na jehož základě seřadí nalezené body zájmu.

Pokud nastane situace, že by se ve výsledku mohly objevit body zájmu se stejným rankem, pak tyto body seřadí ještě podle počtu zadaných slov a počtu slov v názvech bodů zájmu.

Vzorec pro výpočet ranku za každé slovo v zadaném textu je zobrazen v rovnici 2, kde w je jedno slovo v hledaném textu, n udává, jestli je w číslo ($n = 1$) nebo není číslo ($n = 0$), c_{key} je počet bodů zájmu asociovaných s daným klíčem key, c_{max} je maximální počet bodů zájmu asociovaných s libovolným klíčem obsaženým mezi vyhledanými uzly.

Měření	Před úpravou	Po úpravě
Vyhledání „Brno“	40 ms	40 ms
Vyhledání „Ostrava“	114 ms	108 ms
Vyhledání „Moravská“	370 ms	362 ms
Vyhledání „Moravská Ostrava“	478 ms	464 ms

Tabulka 1: Rychlost vyhledávání po úpravě výpočtu Levenshteinovy vzdálenosti

$$rank = n * 10 + (1 - n) * [10 - 3 * (c_{key}/c_{max})] * [(|w| - d_L(w, key))/|w|] \quad (2)$$

2.2.2 Optimalizace vyhledávání

Profilováním kódu fulltextového vyhledávání bylo zjištěno, že poměrně velká část výpočetního času je využita k výpočtu Levenshteinovy vzdálenosti. Dále bylo zjištěno, že v rámci metody výpočtu Levenshteinovy vzdálenosti je nejdelší operace alokace paměti pro proměnné.

Celý výpočet Levenshteinovy vzdálenosti byl proto přesunut do samostatného objektu, který je vytvářený pouze jedenkrát na počátku vyhledávání. V tabulce 1 jsou výsledky měření rychlosti vyhledávání před úpravou a po úpravě výpočtu Levenshteinovy vzdálenosti. Úprava nepřinesla zásadní zrychlení vyhledávání.

Další úprava se snaží optimalizovat strukturu uchovávající děti uzlu. Vhodná volba struktury může ovlivnit velikost celého fulltextového indexu v paměti RAM a rychlost samotného vyhledávání. Děti uzlu jsou asociativní množina, kde klíče jsou datového typy char a hodnoty jsou instance uzlů. Bude měřena rychlost vyhledávání a velikost indexu v paměti RAM pro tyto způsoby uchování dětí:

1. **Pole s prázdnými elementy:** jednotlivé děti jsou uchovány v poli, k jednotlivým uzlům se přistupuje přímo na základě hodnoty klíče, složitost vyhledání je $O(1)$. Počet prvků pole je dán vztahem $k_{max} - k_{min} + 1$, kde k_{min} je hodnota minimálního klíče a k_{max} je hodnota maximálního klíče. Výhoda tohoto způsobu uchování je vysoká rychlost vyhledání, nevýhoda je větší paměťová náročnost v případě nevyužití všech možných klíčů nacházející se mezi k_{min} a k_{max}.

2. **Pole bez prázdných elementů:** jednotlivé děti jsou uchovány v poli, k jednotlivým uzlům se přistupuje sekvenčně, složitost vyhledání je $O(n)$. Počet prvků pole je roven počtu dětí. Výhoda tohoto způsobu uchování je optimální využití paměti RAM, nevýhoda je rychlost vyhledávání.

3. **Seřazené pole:** jednotlivé děti jsou uchovány v poli, k jednotlivým uzlům se přistupuje binárním vyhledáváním, složitost vyhledání je $O(log(n))$. Počet prvků pole je roven počtu dětí. Výhoda tohoto způsobu uchování je optimální využití paměti RAM a rozumná rychlost vyhledávání.

Algoritmus 2.1 Fulltextové vyhledávání bodů zájmu

Vstup: Hledaný text - $text$; maximální počet vyhledaných bodů zájmu - $limit$; kořenový uzel trie, která obsahuje jednotlivé body zájmu, jejichž klíče mají odstraněnou diakritiku a všechny písmena malé - $root$.
Výstup: Množina bodů zájmu odpovídající hledanému textu, seřazených podle podobnosti hledaného textu a názvů bodů zájmu.
Kroky algoritmu:

$words$ = rozděl vstupní $text$ na jednotlivé slova, odstraň diakritiku a změň všechny písmena na malá;
$values$ = nový prázdný seznam nalezených bodů zájmu a jejich ranků (podle ranku budou seřazeny výsledné nalezené body zájmu);
$results$ = nový prázdný slovník asociací $(word : nodes)$, bude obsahovat uzly odpovídající danému slovu;
for $word$ **in** $words$ **do** Vyhledání bodů zajmu pro jednotlivá slova
    if $word$ je číslo **then**
        $values.AddAll($vyhledej přesně všechny body zájmu v trii $root$ a přiřaď každému rank $10)$;
    else
        Maximální Levenshteinova vzdálenost pro nepřesné hledání
        $maxLd = (word.Length <= 8\ ?\ word.Length >> 1 : 4)$;
        $results.AddAll(word : root.Search(word, maxLd))$;
    end if
end for
Najdi maximální počet bodů zájmu ze všech uzlů trie obsažených v slovníku $results$;
$maxValuesCount =$ (from r in $results.Values$ select $r.ValuesCount$).Max();
for $(word, nodes)$ **in** $results$ **do** Výpočet ranků pro jednotlivé výsledky
    Znevýhodnění (0 až 3 body) slov, které se vyskytují nejčastěji.
    $handicap = (nodes.PoisCount()/maxValuesCount) * 3$;
    for $(node, levenshtein)$ **in** $nodes$ **do** Průchod uzly každého výsledku
        for poi **in** $node$ **do** Průchod body zájmu každého uzlu
            $shoda = (word.Length - levenshtein)/word.Length$;
            Připočítání bodů za shodu mezi zadaným slovem a slovem a klíčem uzlu;
            $poi.rank = poi.rank + shoda * (10 - handicap)$;
            $values.Add(poi)$;
        end for
    end for
end for
return Vyber z $values$ maximálně $limit$ bodů zájmu s nejvyšším rankem, má-li více bodů stejný rank, pak tyto body seřaď podle shody počtu slov v zadaném textu $text$ a počtu slov v názvu bodu zájmu.

	Pole $O(1)$	**Pole** $O(n)$	**Pole** $O(log(n))$	**Hashovací tabulka**
Načtení ze souboru	2146 ms	2150 ms	2158 ms	2074 ms
Paměť RAM	61,04 MB	59,96 MB	63,04 MB	71,82 MB
Vyhledávání				
„Brno“	41 ms	41 ms	42 ms	49 ms
„Ostrava“	117 ms	110 ms	120 ms	124 ms
„Moravská“	398 ms	375 ms	400 ms	397 ms
„Moravská Ostrava“	507 ms	469 ms	510 ms	498 ms
„17 633“	3 ms	3 ms	4 ms	3 ms
„17. Listopadu 633“	363 ms	340 ms	365 ms	363 ms

Tabulka 2: Rychlost vyhledávání pro jednotlivé způsoby uchovávání dětí

4. **Hashovací tabulka:** jednotlivé děti jsou uchovány v hashovací tabulce, složitost vyhledání je $O(1)$. Výhoda tohoto způsobu uchování dobrá rychlost vyhledávání a operace vkládání.

Tabulka 2 zobrazuje rychlost vyhledávání a paměťové nároky pro jednotlivé způsoby uchovávání dětí. Nejlepších výsledků bylo dosaženo s polem bez prázdných elementů, se kterým měl fulltextový index nejmenší paměťové nároky a nejrychlejší vyhledávání.

Při fulltextovém vyhledávání algoritmus nejprve najde všechny uzly trie odpovídající jednotlivým slovům zadaného textu v maximální Levenshteinově vzdálenosti určené na základě délky hledaného slova. Vyhledání těchto uzlů se provádí algoritmem nepřesného vyhledávání v trii. Nalezených uzlů může být zbytečně velký počet a některé z nich neovlivní celkový výsledek (vzhledem k zadanému maximální počtu vyhledaných bodů zájmu). Čím přesněji (bezchybněji) uživatel zadá hledaný text, tím spíš budou zajímavé pouze ty uzly, jejichž Levenshteinova vzdálenost od hledaného slova bude nižší.

Tohoto faktu využívá další optimalizace, která upravuje způsob vyhledávání následujícím způsobem:

1. Nepřesné vyhledávání v trii vrací jednotlivé odpovídající uzly v pořadí od nejnižší Levenshteinovy vzdálenosti mezi klíčem uzlu a hledaným slovem.

2. Při průchodu výsledku pro nepřesné vyhledávání se neprochází všechny uzly, ale projdou se vždy pouze uzly s nejnižší Levenshteinovou vzdáleností, a pokud je nalezen minimálně jeden bod zájmu, který odpovídá všem zadaným slovům, pak je hledání ukončeno. Pokud bod zájmu nalezen není, projdou se uzly s větší Levenshteinovou vzdáleností a opět se zkontroluje, jestli byl nalezen minimálně jeden bod zájmu odpovídající všem zadaným slovům. Takto se pokračuje, dokud jsou uzly k prohledání.

Tyto úpravy zajistí, že není nutné procházet tolika uzly trie, ale prochází se tak dlouho, dokud není nalezen hledaný bod zájmu. Protože samotné procházení trie při nepřesném vyhledávání v trii je časově náročná operace, snížení počtu navštívených uzlů trie by mělo zvýšit rychlost vyhledávání.

Hledaný test	Neseřazený průchod	Seřazený průchod
Vyhledání „Aš“	7,5 ms	0,5 ms
Vyhledání „Ostrava“	104 ms	1,5 ms
Vyhledání „Moravská“	350 ms	0,7 ms
Vyhledání „Moravská Ostrava“	453 ms	2,2 ms
Vyhledání „Moravská Oxtxaxa“	439 ms	137 ms
Vyhledání „Oxtxaxa“	95 ms	134 ms
Vyhledání „Moravs Oxtxaxa“	183 ms	283 ms

Tabulka 3: Změna rychlosti vyhledávání při seřazeném průchodu trií

Úprava nepřesného vyhledávání v trii, tak, aby vracelo jednotlivé odpovídající uzly v pořadí od nejnižší Levenshteinovy vzdálenosti mezi klíčem uzlu a hledaným slovem, je provedena s využitím seřazeného seznamu uzlů. Do seznamu se vkládají uzly s vypočítanou Levenshteinovou vzdáleností mezi klíčem uzlu a hledaným slovem. Ze seznamu se vyberou ty uzly, které mají nejnižší Levenshteinovu vzdálenost a prohledají se jejich potomci a opět se vloží do seznamu. To se opakuje, dokud přibývají uzly s menší Levenshteinovu vzdáleností. Jednotlivé uzly v pořadí od nejnižší Levenshteinovy vzdálenosti se začnou iterativně vracet jako výsledek nepřesného hledání v momentě, kdy je nalezen uzel s nulovou Levenshteinovou vzdáleností nebo neexistují žádné uzly s menší Levenshteinovou vzdáleností.

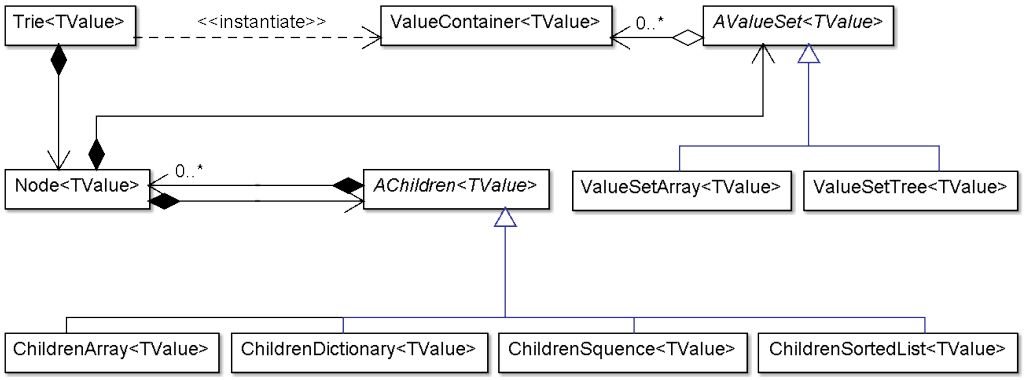
Tato úprava nepřesného vyhledávání v trii způsobí vetší režii, především s údržbou seřazeného seznamu uzlů. Tato vyšší režie je kompenzována v navazujícím kroku fulltextového vyhledávání, kdy je možné projít menší počet uzlů, než je nalezen výsledek.

V tabulce 3 je srovnání časů vyhledávání oběma způsoby. Při vyhledávání „Aš“, „Ostrava“ a „Moravská“ se projevil vliv snížení počtu navštívených uzlů. Protože tyto slova byla zadána přesně, byl rychle nalezen uzel s nulovou Levenshteinovou vzdáleností. Při vyhledávání „Moravská Ostrava“ se navíc projevila optimalizace, kdy vyhledávání končí, jakmile je nalezen bod zájmu odpovídající všem zadaným slovům. Při vyhledávání „Oxtxaxa“ a „Moravs Oxtxaxa“ optimalizace průchodu nepomohla, protože zadaná slova neodpovídají žádným klíčům v trii a naopak se projevila vyšší režie s údržbou seřazeného seznamu uzlů. Při vyhledání fráze „Moravská Oxtxaxa“ se projevil přínos rychlého vyhledání slova „Moravská“, tak i nevýhoda pomalejšího vyhledávání nepřesně zadaných slov, ale i přesto byl čas vyhledání kratší, oproti původnímu způsobu vyhledávání.

Na obrázku 5 je diagram tříd pro fulltextové vyhledávání. Datová struktura trie je implementována jako generická, aby byla univerzální a bylo ji možné využít i pro vyhledávání jiných dat než bodů zájmu.

2.3 Další datové struktury

Pro vyhledáván řetězců lze použít i jiné datové struktury. V této kapitole je uveden stručný přehled takových struktur.



Obrázek 5: Diagram tříd trie

2.3.1 Ternární strom

Ternární strom [5] je další datovou strukturou, kterou je možné použít pro vyhledávání řetězců. Každý uzel ternárního stromu má tři děti, při vyhledávání se porovná znak uzlu se znakem hledaného řetězce; pokud je znak řetězce menší než znak uzlu, pokračuje se levým potomkem; pokud je znak řetězce větší než znak uzlu, pokračuje se pravým potomkem; pokud je znak řetězce roven znaku uzlu, pokračuje se prostředním potomkem a vyhledává se následující znak řetězce.

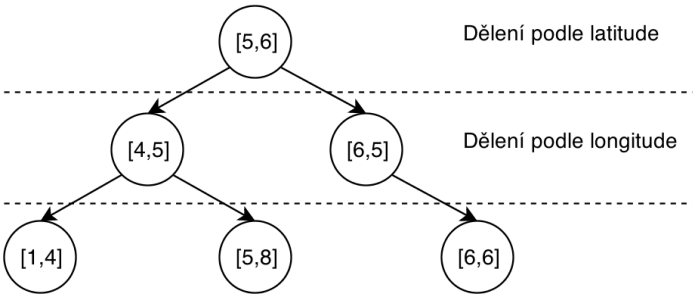
Podle [5] je výhodou ternárních stromů, oproti triím, menší paměťová náročnost a naopak výhodou trií je vyšší rychlost vyhledávání. Protože implementované fulltextové vyhledávání bude využito ve webové službě, má vyšší prioritu rychlost vyhledávání. Větší paměťové nároky trií budou sníženy využitím komprimované trie (kapitola 2.1) a vhodnou volbou způsobu uložení dětí uzlů trie (kapitola 2.2.2).

2.3.2 Sufixový strom

Sufixový strom [14] je trie, která obsahuje všechny sufixy řetězce, kromě sufixu nulové délky. Sufixový strom například umožňuje rychlé nalezení řetězců na základě znalosti jejich sufixu nebo prefixu. Protože sufixový strom obsahuje všechny sufixy řetězce, je jeho paměťová složitost větší než paměťová složitost trie.

2.3.3 Hashovací tabulka

Hashovací tabulka [5] je datová struktura pro uchování množiny asociací. Hodnoty ukládá v poli, pozice hodnot v poli je dána na základě hashovací funkce. Její výhodou je složitost vyhledávání $O(1)$ v ideálním případě. Pro účely nepřesného vyhledávání, které je potřeba pro vytvoření fulltextového indexu se nehodí, protože jednotlivé hodnoty jsou v tabulce uspořádány na základě hashovací funkce, ne na základě podobnosti klíčů, takže při nepřesném vyhledávání by se musely projít všechny hodnoty v tabulce obsažené.



Obrázek 6: Ukázka dvourozměrného k-d stromu

2.4 K-d strom

K-d strom [6] je datová struktura používaná pro organizaci bodů v k-rozměrném prostoru. K-d strom je binární vyhledávací strom. Pro organizaci bodů zájmu bude použita dvourozměrná varianta. K-d strom lze využít pro vyhledávání nejbližších bodů zájmu k zadaným souřadnicím a vyhledávání všech bodů zájmu v zadané oblasti.

Definice 2.2 *„Strom se nazývá dokonale vyvážený, jestliže pro každý uzel stromu platí, že počet uzlů v jeho levém a pravém podstromu se liší nejvýše o jeden.“ [5, str. 149]*

K-d strom dělí v každém uzlu, který není listem, body zájmu na dvě části. Dělení probíhá v každé úrovni střídavě podle zeměpisné šířky a zeměpisné délky, například v první úrovni se body zájmu rozdělí podle zeměpisné šířky, v následující podle zeměpisné délky, atd.

Dělení může být na dvě stejně velké části nebo podle některého bodu zájmu. Dělení na dvě stejně velké části umožní v ideálním případě dosáhnout dokonale vyváženého stromu, proto bude použito. Při dělení na dvě stejně velké části se body zájmu rozdělí tak, že levý podstrom obsahuje body zájmu s menší nebo shodnou hodnotou zeměpisné šířky / délky (podle úrovně daného uzlu) jako bod $pivot$ a pravý podstrom obsahuje body zájmu s hodnotou vyšší [17]. Na obrázku 6 je k-d strom obsahující body {[1, 4], [5, 8], [6, 6], [4, 5], [6, 5], [5, 6]}.

2.4.1 Konstrukce dvourozměrného k-d stromu

K-d strom je konstruován algoritmem 2.2. Vstupem algoritmu je množina bodů. Algoritmus seřadí vstupní množinu bodů podle zeměpisné šířky nebo délky (podle aktuální hloubky rekurze), určí prostřední prvek, který dělí vstupní množinu na dvě podmnožiny (levá podmnožina obsahuje prvky menší nebo rovny prostřednímu prvku a pravá podmnožina obsahuje prvky větší) a rekurentně provede konstrukci k-d stromu pro pravou a levou podmnožinu.

Algoritmus 2.2 Vytvoření k-d stromu

Vstup: Množina bodů zájmu P, každý bod zájmu obsahuje zeměpisnou šířkou ($latitude$) a délku ($longitude$), hloubka rekurze $deep$
Výstup: k-d strom obsahující všechny body zájmu z množiny P

Kód:

$node$ = vytvoř nový prázdný uzel;
Určení podle které souřadnice se dělí v aktuální úrovni.
$divide$ = $(deep \bmod 2) == 0$? $latitude$: $longitude$;
Seřaď množinu P podle souřadnice $divide$;
$pivotIndex$ = urči prostřední bod množiny P;
$node.pivot = P[pivotIndex]$;
Pokud obě podmnožiny mají minimálně jeden bod
if $|P[1..pivotIndex - 1]| > 0 \wedge |P[pivotIndex + 1..|P|]| > 0$ **then**
    $node.left$ = rekurentně vytvoř k-d strom z $P[1..pivotIndex - 1]$ a $deep + 1$;
    $node.right$ = rekurentně vytvoř k-d strom z $P[pivotIndex + 1..|P|]$ a $deep + 1$;
end if
return $node$;

2.4.2 Vyhledání bodů zájmu v dané oblasti

Vyhledání bodů zájmu v dané oblasti umožňuje vyhledávat na základě zadaného rozsahu zeměpisné délky, šířky a volitelně kategorie bodů zájmu. Kategorie mohou dělit body zájmu například na stravování, turistické cíle, zábavu a podobně.

Vyhledání bodů zájmu v dané oblasti popisuje algoritmus 2.3. Algoritmus prochází stromem od kořene, zjistí, zdali aktuální uzel leží celý v zadaném rozsahu a pokud ano, potom všechny hodnoty obsažené v uzlu a jeho potomcích přidá do množiny nalezených bodů zájmu. Pokud aktuální uzel svými hranicemi pouze překrývá část rozsahu, pak se projdou jeho děti a pokud jeho prostřední bod leží v zadaném rozsahu, přidá se do množiny nalezených bodů zájmu.

Před přidáváním libovolného bodu do množiny nalezených bodů zájmu je provedena kontrola, zdali daný bod patří do požadované skupiny bodů zájmu, pokud nepatří, nebude přidán.

2.4.3 Vyhledání nejbližších bodů zájmu na daných souřadnicích

Vyhledání nejbližších bodů zájmu na daných souřadnicích umožňuje vyhledávat na základě zadané zeměpisné souřadnice (středu vyhledávání). Vyhledávání najde n nejbližších bodů zájmu k daným souřadnicím. Volitelně může být zadána kategorie bodů zájmu pro vyfiltrování.

Zdrojový kód algoritmu pro vyhledání nejbližších bodů zájmu je uveden ve výpisu 3. Vstupem metody je poloha středu vyhledávání, limit počtu vyhledaných nejbližších bodů

Algoritmus 2.3 Vyhledání bodů zájmu v dané oblasti a dané kategorie v k-d stromu

Vstup: K-d strom - $root$ - obsahující množinu bodů zájmu - P; rozsah zeměpisné délky a šířky - $range$; množina kategorií bodů zájmu - C
Výstup: $\{p \in P : p \in range \wedge p \in C\}$
Kód:

```
queue = nová prázdná fronta uzlů k prohledání;
queue.Enqueue(root);
result = nový prázdný seznam nalezených bodů zájmu;
while |queue| > 0 do Dokud fronta obsahuje prvky
    node = queue.Dequeue();
    if range.Contains(node.bounds) then Pokus rozsah obsahuje hranice uzlu
        Přidání do výsledku všech bodů zájmu v daných kategoriích
        result = result ∪ {p ∈ node.Values() : p ∈ C};
    else range.IsIntersection(node.bounds) Pokud se rozsah překrývá s oblastí uzlu
        if node.left! = null then Pokud má levé dítě
            queue.Enqueue(node.left);
        end if
        if node.right! = null then Pokud má pravé dítě
            queue.Enqueue(node.right);
        end if
        Pokud pivot leží v zadaném rozsahu a má správnou kategorii
        if node.pivot ∈ range ∧ node.pivot ∈ C then
            result = result ∪ {node.pivot};
        end if
    end if
end while
return result;
```

zájmu a filtr kategorií. Na obrázku 7 je znázorněn průchod stromem při vyhledávání nejbližších bodů k bodu [5, 7].

Na řádku 3 je definovaný setříděný seznam nejbližších bodů zájmu, v tomto seznamu se udržují nejbližší nalezené body zájmu, maximální délka seznamu je omezena limitem počtu nalezených bodů zájmu. Na řádku 4 je definovaná oblast, u které je předpoklad, že obsahuje některý z nejbližších bodů zájmu. Tato oblast na počátku zahrnuje vše. Na řádku 5 je definován seznam uzlů, které se musí zkontrolovat na výskyt nejbližších bodů zájmu.

Cyklus na řádku 7 prochází stromem postupně do hloubky (na obrázku 7 je průchod vyznačen zelenými šipkami), dokud nenarazí na list, a vždy volí dítě, které se nachází na stejné polorovině jako střed vyhledávání. Opačné dítě se přidá do seznamu uzlů $others$, které se musí později zkontrolovat. V průběhu průchodu stromem se udržuje seznam nejbližších bodů zájmu (řádek 12), pomocí metody $MaintainNearest$, která zjistí, jestli se má bod zájmu $n.Pivot$ vložit mezi nejbližší body zájmu na základě jeho vzdálenosti ke

středu vyhledávání. Pokud dojde ke vložení daného bodu zájmu mezi nejbližší, metoda upraví podezřelou oblast vyhledávání, $check$, tak, aby zahrnovala oblast danou nejvzdálenějším bodem z nejbližších nalezených bodů. Metoda uzlu $GetByLocation$ (na řádku 14) vrátí dítě, které obsahuje danou lokaci. Metoda uzlu $GetOtherByLocation$ (na řádku 9) vrátí dítě, které neobsahuje danou lokaci

Cyklus na řádku 17 projde všechny body zájmu v listu nalezeném v předchozím cyklu a pokusí se je zařadit do seznamu nejbližších. V případě vyhledávání ve stromu na obrázku 7 se pokusí zařadit bod $[5, 8]$.

Na řádku 22 je cyklus, který postupně prochází seznam uzlů, které se musí zkontrolovat (červené šipky na obrázku 7). Seznam se prochází od konce, tedy od uzlů, které jsou nejvíce vnořeny ve struktuře stromu, tím je zajištěna kontrola uzlů, obsahující menší oblast, dříve, než uzlů obsahující oblast větší. Kontrola uzlů probíhá pouze, pokud překrývají oblast, která může obsahovat nejbližší body zájmu. Metoda $CheckSubtree$ na řádku 28 zkontroluje podstrom daného uzlu, prochází pouze potomky, kteří překrývají podezřelou oblast.

Nakonec dojde na řádku 32 k ukončení metody a vrácení všech nalezených nejbližších bodů zájmu.

```
1  public IEnumerable<TValue> GetNearest(Location location, uint countLimit, IFilter<TValue> filter)
2  {
3    SortedSet<ValueDistance> nearest = new SortedSet<ValueDistance>();
4    Range check = Range.MaxValue;
5    LinkedList<Node<TValue>> others = new LinkedList<Node<TValue>>();
6    Node<TValue> n = GetRoot();
7    while(n.IsNotLeaf)
8    {
9      others.AddLast(n.GetOtherByLocation(location));
10     if ( filter .Contains(n.Pivot.Value))
11     {
12       MaintainNearest(nearest, location, n.Pivot, countLimit, ref check);
13     }
14     n = n.GetByLocation(location);
15   }
16
17   foreach (var value in n.Where(value => filter.Contains(value.Value)))
18   {
19     MaintainNearest(nearest, location, value, countLimit, ref check);
20   }
21
22   while (others.Count > 0)
23   {
24     n = others.Last.Value;
25     others.RemoveLast();
26     if (n.Bounds.IsIntersection(check))
27     {
28       CheckSubtree(n, location, ref check, nearest, countLimit, filter );
29     }
30   }
31
32   return from i in nearest select i.Value.Poi;
```

```
33 }
```

Výpis 3: Kód metody pro vyhledání nejbližších bodů zájmu na daných souřadnicích

2.4.4 Ukázka vyhledání nejbližších bodů zájmu

Je dán k-d strom zobrazený na obrázku 7 a souřadnice $[5, 7]$, ke kterým se musí vyhledat dva nejbližší body. Algoritmus provede následující kroky:

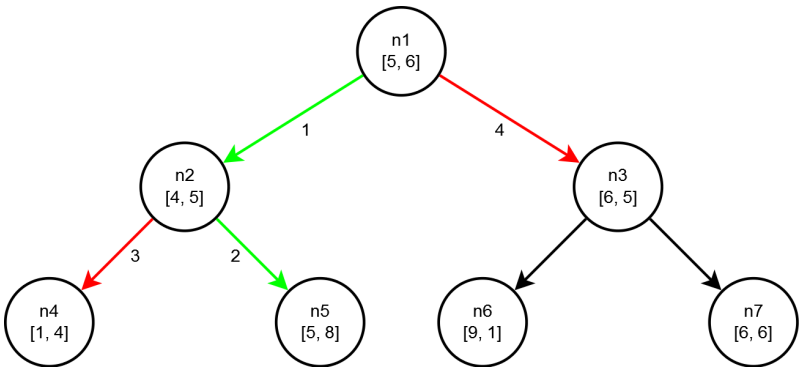
1. Inicializují se počáteční proměnné.
   $nearest = \emptyset$
   $check =< [-\infty, -\infty], [\infty, \infty] >$
   $others = \emptyset$

2. Prochází se do hloubky, aktuální uzel je $n1$.
   $nearest = \{([5, 6], 1)\}$
   $check =< [-\infty, -\infty], [\infty, \infty] >$
   $others = \{n3\}$

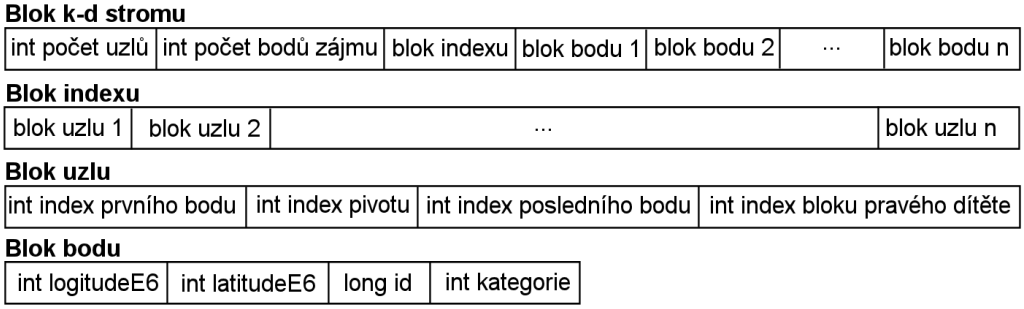
3. Prochází se do hloubky, aktuální uzel je $n2$.
   $nearest = \{([5, 6], 1), ([4, 5], \sqrt{5})\}$
   $check =< [5 - \sqrt{5}, 6 - \sqrt{5}], [5 + \sqrt{5}, 6 + \sqrt{5}] >$
   $others = \{n3, n4\}$

4. Prochází se do hloubky, aktuální uzel je $n5$.
   $nearest = \{([5, 6], 1), ([5, 8], 1)\}$
   $check =< [5 - 1, 8 - 1], [5 + 1, 8 + 1] >$
   $others = \{n3, n4\}$

5. Prochází se zpět, aktuální uzel je $n4$, oblast uzlu se nepřekrývá s oblastí $check$.
   $nearest, check$: beze změny
   $others = \{n3\}$



Obrázek 7: Ilustrace průchodu stromem při vyhledávání nejbližších bodů k bodu [5, 7]



Obrázek 8: Formát uložení k-d stromu na disku

6. Prochází se zpět, aktuální uzel je $n3$, oblast uzlu se nepřekrývá s oblastí $check$.
   $nearest, check$: beze změny
   $others = \emptyset$

7. Vyhledávání ukončeno, výsledkem jsou body v množině $nearest$.

2.4.5 Formát ukládání k-d stromu

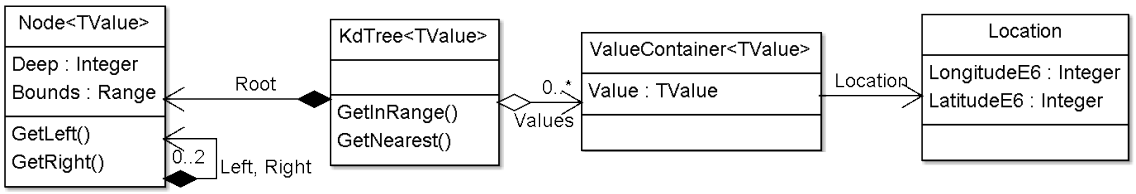
Aby se jednou vytvořený k-d strom nemusel opakovaně vytvářet při restartu webové služby, je vytvořen formát pro jeho uložení do souboru na disk. Z takto vytvořeného souboru lze rychle načíst do paměti celý strom. Na obrázku 8 je zobrazena struktura formátu souboru k-d stromu.

Základní částí je blok k-d stromu, který obsahuje informaci o počtu uzlů, dále o počtu bodů zájmu, poté blok indexu následovaný jednotlivými bloky bodů (pokud by strom neobsahoval žádné body, pak zde žádné bloky nebudou).

Blok indexu obsahuje bloky jednotlivých uzlů, počet bloků uzlů je dán počtem uzlů ve stromu. Bloky uzlu jsou zapsány v pořadí preorder [5]. Každý blok uzlu obsahuje index prvního bloku bodu patřícího do uzlu, index posledního bloku bodu patřícího do uzlu a index bloku bodu, který je pivotem uzlu. Dále index bloku uzlu pravého potomka. Index bloku uzlu levého potomka není potřeba ukládat, protože díky uspořádání preorder, tento blok následuje přímo za aktuálním blokem.

Formát k-d stromu v paměti RAM je velice podobný formátu souboru na disku. Blok indexu je realizován polem celých čísel, každý blok uzlu obsahuje čtyři čísla, celková velikost pole tedy je $4 * n$ celých čísel, kde n je počet uzlů ve stromu. Všechny body ve stromu jsou uloženy také v jediném poli, indexy v poli odpovídají indexům bloků bodů v souboru. Veškerá data, která tvoří strom, jsou uložena ve dvou polích. Při načítání stromu ze souboru stačí provést dvě alokace polí, jejichž velikost je předem známa, a alokace pro jednotlivé body zájmu (odpovídá blokům bodů zájmu).

Protože je celá struktura stromu uložena ve dvou polích, odpadá nutnost vytváření instancí jednotlivých uzlů stromu. K zjištění veškerých informací o uzlu stačí znát index do pole indexů (tím zjistíme, které všechny body patří do uzlu, jeho pivot a jestli má potomky) a hloubku zanoření (tím je dána osa pivotu, která dělí jednotlivé body).



Obrázek 9: Diagram tříd k-d stromu

Na obrázku 9 je diagram tříd k-d stromu. Datová struktura k-d stromu je implementována jako generická, aby byly univerzální a bylo je možné využít i pro vyhledávání jiných dat než bodů zájmu.

2.4.6 Test rychlosti vyhledávání v k-d stromu

Byly provedeny testy rychlosti jednotlivých způsobů vyhledávání v k-d stromu. Při testech byly v k-d stromu vloženy body zájmu v ČR. Počet adres byl 1 020 344 a počet ostatních bodů zájmu (například restaurace a kultura) byl 107 270.

První test měřil rychlost vyhledávání bodů zájmu v zadané oblasti. Výsledky testu jsou v tabulce 4. Měření proběhlo pro tyto oblasti[1]:

1. **<[0°; 0°], [5°; 5°]>**: oblast pro kterou strom neobsahuje žádné body zájmu.
2. **<[49,834049°; 18,160130°], [49,834049°; 18,160130°]>**: oblast obsahující jediný bod zájmu - VŠB-TUO.
3. **<[49,830329°; 18,159366°], [49,839833°; 18,159366°]>**: oblast v okolí VŠB-TUO, obsahuje 42 bodů zájmu.
4. **<[49,815975°; 18,142588°], [49,845944°; 18,198819°]>**: oblast zahrnující celou Porubu v Ostravě, zahrnuje 301 bodů zájmu.
5. **<[49,742148°; 18,148311°], [49,871746°; 18,449843°]>**: oblast zahrnující celou Ostravu, zahrnuje 2 209 bodů zájmu.
6. **<[0°; 0°], [90°; 180°]>**: oblast zahrnující všechny body ve stromu zahrnuje 107 270 bodů zájmu.

Druhý test měřil rychlost vyhledání nejbližších bodů zájmu (bez filtrace kategorií a s filtrováním kategorie vzdělání) a adres k zadaným souřadnicím. Výsledky testu jsou v tabulce 5. Vyhledávání probíhalo postupně s omezením maximálního počtu vyhledávaných nejbližších bodů na jeden, deset a sto bodů zájmu. Jako střed vyhledávání byly zvoleny tyto lokace:

1. **[49,508065°; 18,370670°]**: hora Smrk v Beskydech, oblast řídce pokrytá body zájmu.

[1]Souřadnice jsou zapsané v formátu [zeměpisná šířka; zeměpisná délka]

Oblast	**Počet bodů**	**Doba vyhledávání**
<[0°; 0°], [5°; 5°]>	0	0,02 ms
<[49,834049°; 18,160130°], [49,834049°; 18,160130°]>	1	0,021 ms
<[49,830329°; 18,159366°], [49,839833°; 18,159366°]>	42	0,024 ms
<[49,815975°; 18,142588°], [49,845944°; 18,198819°]>	301	0,129 ms
<[49,742148°; 18,148311°], [49,871746°; 18,449843°]>	2209	0,416 ms
<[0°; 0°], [90°; 180°]>	107 270	12,654 ms

Tabulka 4: Rychlost vyhledávání bodů zájmu v zadané oblasti

Nastavení		**Doba vyhledávání**		
Střed vyhledávání	Počet bodů	adresy	vše	vzdělání
[49,508065°; 18,370670°]	1	0,281 ms	0,053 ms	0,401 ms
	10	0,351 ms	0,205 ms	0,811 ms
	100	0,913 ms	0,856 ms	2,594 ms
[49,834049°; 18,160130°]	1	0,066 ms	0,057 ms	0,033 ms
	10	0,113 ms	0,088 ms	0,126 ms
	100	0,606 ms	0,404 ms	1,962 ms
[[49,834933°; 18,280474°]	1	0,095 ms	0,056 ms	0,051 ms
	10	0,153 ms	0,101 ms	0,244 ms
	100	0,608 ms	0,527 ms	1,715 ms

Tabulka 5: Rychlost vyhledávání nejbližších bodů zájmu k zadaným souřadnicím

2. **[49,834049°; 18,160130°]**: VŠB-TUO, oblast středně hustě pokrytá body zájmu.
3. **[49,834933°; 18,280474°]**: Stodolní ulice v Ostravě, oblast hustě pokrytá body zájmu.

V obou testech byla doba vyhledávání ve většině měření kratší než 1 ms, tento výsledek je uspokojivý, proto lze testované algoritmy považovat za dostatečně výkonné pro nasazení ve webové službě.

3 Nasazení implementovaného řešení pomocí webové služby

Webová služba [23] je software navržený k podpoře spolupráce zařízení přes síť. Má přesně popsané rozhraní ve strojově zpracovatelném formátu, podle kterého probíhá komunikace ostatních zařízení s webovou službou, typicky pomocí HTTP s XML serializací.

Knihovna pro vyhledávání bodů zájmu a vyvíjený framework pro směrování vozidel je implementován pro prostředí .NET, proto i samotná služba je vyvíjena pro stejné prostředí. Webové služby ASP.NET umožňují komunikaci s zařízeními prostřednictvím XML a JSON zpráv. JSON formát je výhodný například pro komunikace s zařízeními využívající operační systém Android, protože obsahují nativní podporu pro zpracování JSON formátu.

3.1 Funkce webové služby

Smyslem vyvíjené webové služby je nabídnout škálu funkcí, která obsáhne všechny důležité funkce potřebné ve směrovacích aplikacích. Jednotlivé funkce služby pak mohou využívat další aplikace, které je využijí podle svého zaměření.

Webová služba nabídne funkce pro vyhledání dostupnosti, cesty mezi dvěma body, nejbližších bodů zájmu v okolí, přibližné adresy na daných souřadnicích, fulltextové vyhledání bodů zájmu a bodů zájmu v oblasti vymezené rozsahem souřadnic.

Funkce vyhledání dostupnosti umožňuje vypočítat dojezd ze zadaných počátečních bodů, včetně možnosti zvolit nedostupné oblasti (oblasti, které jsou neprůjezdné). Služba také obsahuje seznam hasičských stanic a informace o zatopených oblastech při stoleté, dvacetileté a pětileté vodě, takže lze nechat vypočítat například mapu pokrytí hasičskými jednotkami tak, aby dojezd hasičů nepřekročil dvacet minut.

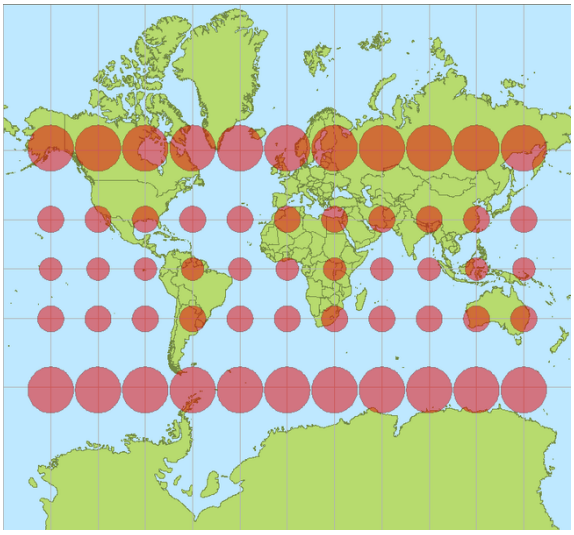
Další využití může být například v aplikaci, která umožní uživateli zjistit si, kam až dojede od svého domu v zadaném časovém limitu. Protože samotný výsledek vyhledání dostupnosti může být paměťově náročný, především při vyhledávání dostupnosti z všech hasičských stanic, proto je celý výsledek rozdělený do dlaždic a klientovi se průběžně zasílají ty dlaždice, o které si zažádá.

Funkce vyhledání cesty mezi dvěma body vyhledá cestu, vrátí jednotlivé úseky cesty a informaci o délce cesty a odhadované době jízdy. Funkce pro vyhledávání bodů zájmu vrací jednotlivé nalezené body, každý bod obsahuje informace: jméno bodu, popis bodu, souřadnice umístění, typ bodu (například „autobusová zastávka“) a kategorii bodu (například „doprava“).

3.2 Rozdělení mapy na dlaždice

Při vyhledávání dostupnosti je výsledek vracen postupně v jednotlivých dlaždicích, o které si klient žádá. Rozdělení na jednotlivé dlaždice je shodné jako rozdělení na dlaždice u Microsoft Bing map nebo Mapniku (nástroj pro generování mapových podkladů [18]).

Využívá Mercatorovo zobrazení, které má dvě zásadní vlastnosti: zachovává tvar relativně malých objektů, sever a jih jsou vždy nahoře a dole, východ a západ jsou vždy



Obrázek 10: Zkreslení polárních oblastí u Mercatorova zobrazení [15]

vpravo a vlevo. Protože projekce u pólů se blíží nekonečnu, je zobrazovaný rozsah zeměpisné šířky v rozsahu od $-85,05°$ do $85,05°$ [16].

Na obrázku 10 je zobrazeno zkreslení objektů blížících se k pólům. Všechny kruhy na mapě znázorňují stejně velkou plochu země. Lze vidět, že Mercatorovo zobrazení zobrazuje objekty blížící se k pólům větší oproti objektům blíže k poledníku.

Každá dlaždice má velikost 256 x 256 pixelů. Počet dlaždic na mapě závisí na úrovni rozlišení. Počet dlaždic na výšku a šířku mapy je shodný, je dán vztahem $n_x = n_y = 2^d$, kde n_x je počet dlaždic na šířku, n_y je počet dlaždic na výšku a d je úroveň rozlišení. Celkový počet dlaždic je pak dán vztahem $n = (2^d)^2$. Pro identifikaci každé dlaždice proto je potřeba znát úroveň rozlišení a souřadnice x a y.

3.3 Kešování výsledků dostupnosti

Protože samotný výpočet dostupnosti je časově a paměťově náročná operace (především pokud se počítá dostupnost z více počátečních míst), je nutné udržovat si výsledek v paměti tak, aby při každém dotazu klienta na další dlaždici výsledku nebylo nutné vypočítávat celou dostupnost znova. Proto se výsledek dostupnosti uloží do speciální struktury, která data rozdělí do jednotlivých dlaždic a při požadavku na konkrétní dlaždici se pracuje už pouze s relevantní částí dat.

3.4 Proces vyhledání dostupnosti

Pokud chce klient vypočítat dostupnost, pak musí první zavolat metodu $PrepareAccessability$, jejíž parametry jsou všechny údaje nutné k vypočítání dostupnosti, tedy maximální rychlost vozidla, souřadnice počátku vyhledávání (v případě pokud vyhledává dostupnost z jednoho bodu), identifikátory skupin počátečních bodů (po-

kud vyhledává dostupnost z více počátečních lokací, například dostupnost z hasičských stanic), jestli se mají používat pouze úseky průjezdné nákladním vozidlem a identifikátor omezení (v případě omezení například stoletou vodou). Po zavolání $PrepareAccessability$ může klient opakovaně volat metodu $GetAccessabilityTile$, která vrací dostupnost pro jednotlivé dlaždice mapy.

4 Vytvoření aplikace pro OS Android

V této kapitole budou nejprve uvedeny základní informace o OS Android, jeho vlastnosti, zastoupení jednotlivých verzí a výběr verze, pro kterou bude naše aplikace vytvářena. Poté bude detailněji představena vytvářená aplikace.

4.1 Operační systém android

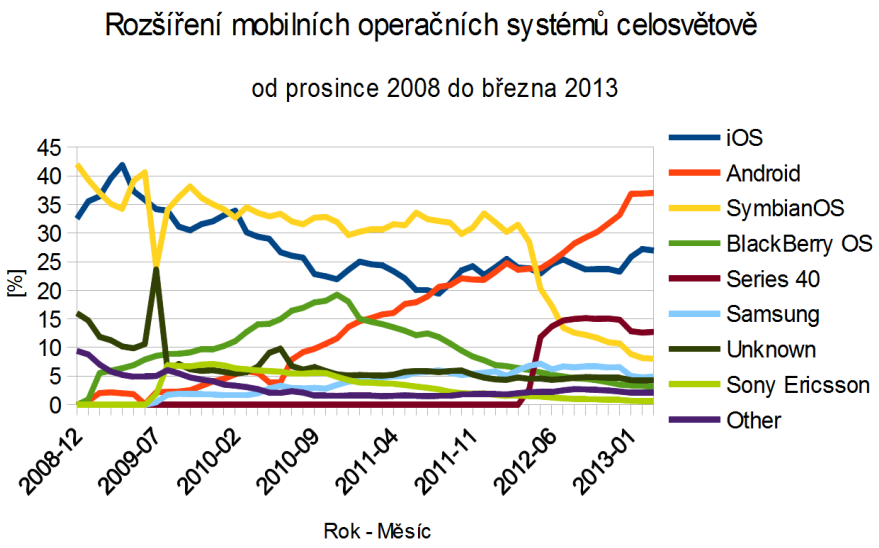
OS Android je postaven na Linuxovém jádře s otevřeným zdrojovým kódem. Je na miliónech mobilních zařízení na celém světě. Díky otevřenému zdrojovému kódu jeho rozšíření rychle roste [21].

Je navržen primárně jako dotekový OS, využívá se především v chytrých telefonech a tabletech, ale objevuje se i v zařízeních jako fotoaparáty, multimediální přehrávače, čtečky knih a dokonce i v lednicích. OS Android byl první, který kombinoval tyto výhody [2]:

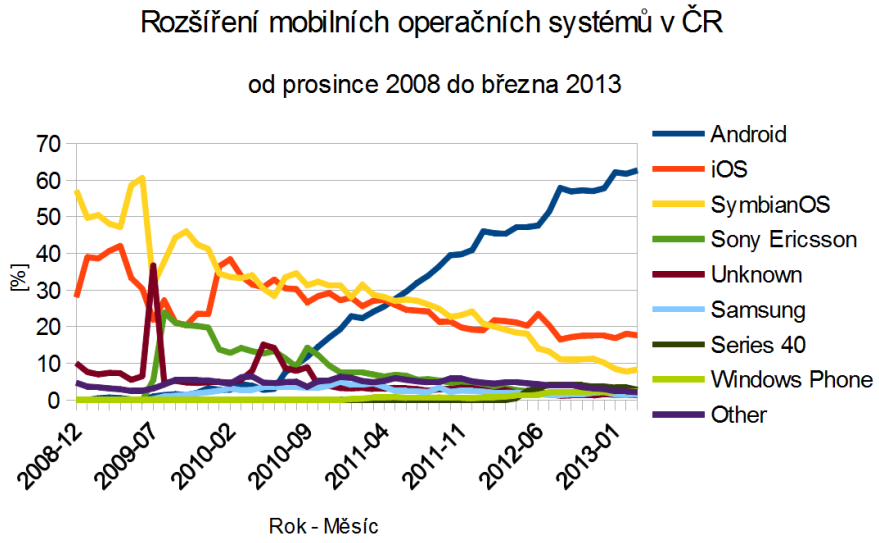
- Bezplatné vývojové prostředí a otevřený zdrojový kód. Protože výrobci telefonů nemusejí platit licenční poplatky a vývojáři mají dostupné bezplatné vývojové prostředí, stal se rychle oblíbený.
- Komponentová architektura inspirovaná internetovými mashup aplikacemi umožňuje nahradit části aplikace vlastními upravenými komponentami.
- Množství vestavěných služeb, jako polohové služby využívající GPS a triangulace pomocí dostupných buněk mobilních operátorů, SQL databáze, prohlížeč, mapová komponenta.
- Automatické řízení životního cyklu aplikace. Jednotlivé běžící aplikace jsou izolované, Android je optimalizovaný pro málo výkonné zařízení s omezeným množstvím paměti.
- Vysoká kvalita grafiky a zvuku. Jemná a vyhlazená 2D vektorová grafika a animace inspirované Flash aplikacemi a 3D grafika akcelerovaná pomocí OpenGL. Podpora multimediálních kodeků H.264, MP3 a AAC.
- Přenositelnost mezi velkým množstvím současného a budoucího hardwaru.

Zastoupení OS Android mezi mobilními OS z dlouhodobého hlediska roste. Podle statistik StatCounter bylo jeho celosvětové zastoupení 37% v březnu 2013 (graf na obrázku 11), zastoupení v České Republice pak dokonce 62,7% (graf na obrázku 12) [22]. Existují různé verze Androidu, při vývoji aplikace je nutné na počátku rozhodnout pro kterou verzi bude vyvíjena.

Volbou starší verze je výhoda větší kompatibility (aplikace pro starší verze lze spustit i na novějších verzích), ale nevýhodou je starší API, které je chudší, může obsahovat chyby, jež jsou v novějších verzích odstraněny, absence některých technologií. Proto je nutné najít kompromis mezi verzí a její penetrací. Z tabulky 6 lze usoudit, že volbou API 10 lze získat podporu 89,3% zařízení (součet penetrací všech API ≥ 10).



Obrázek 11: Zastoupení mobilních operačních systému celosvětově [22]



Obrázek 12: Zastoupení mobilních operačních systému v ČR [22]

Verze	Označení	API	Penetrace
1.6	Donut	4	0.2%
2.1	Eclair	7	2.2%
2.2	Froyo	8	8.1%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7	Gingerbread	10	45.4%
3.1	Honeycomb	12	0.3%
3.2	Honeycomb	13	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.0%
4.1	Jelly Bean	16	12.2%
4.2	Jelly Bean	17	1.4%

Tabulka 6: Zastoupení jednotlivých verzí OS Android 4. 2. 2013 [9]

4.1.1 Distribuce Android aplikací

Primárním distribučním kanálem pro Android aplikace je Google Play, což je internetový obchod, přes který můžou uživatelé androidu instalovat aplikace do zařízení. Aplikace v obchodě mohou být bezplatné i placené.

Zákazníkovi je umožněno nakoupit pouze takové aplikace, které jsou určeny pro jeho verzi Androidu. Při umístění aplikace do obchodu může autor aplikace omezit distribuci pouze pro konkrétní hardwarové konfigurace. Každý měsíc je z Google Play staženo 1,5 miliardy aplikací a počet stále roste. [8]

Dalším způsobem distribuce aplikace pomocí e-mailu. Aplikaci stačí připojit jako přílohu e-mailové zprávy a odeslat na Gmail uživatele. Pokud si uživatel zprávu vyzvedne prostřednictvím Gmail aplikace, bude mu nabídnuta instalace aplikace. Zobrazí se mu však varování, že instaluje aplikaci z neověřeného zdroje. Proto je tento způsob šíření vhodný pouze pro omezenou skupinu uživatelů.

Posledním způsobem je umístění aplikace na vlastní webový server. V tomto případě se uživatelům bude před instalací zobrazovat varování, že instalují aplikaci z neověřeného zdroje. V případě šíření aplikace přes e-mail i přes vlastní webový server, musí mít uživatelé v nastavení telefonu povoleno instalovat aplikace z neznámých zdrojů.

4.1.2 Příprava aplikace pro distribuci

Jakmile je aplikace připravena pro distribuci přes Google Play, musí být provedeno několik kroků, před samotným umístěním aplikace do nabídky Google Play. První je potřeba zkontrolovat název balíčku aplikace, musí být takový, aby byl pro aplikaci vhodný po celou dobu její existence. Změna jména balíčku není později možná.

Další krok je odstranění z kódu volání logování a zakázání ladění aplikace. Také je potřeba z aplikace odstranit všechny testovací soubory a logy, které nejsou nutné pro běh aplikace.

Další krok je vhodná volba ikony aplikace, na kterou uživatel klepne pro spuštění aplikace. Ikona by měla být jednoznačná a nezaměnitelná s ikonami jiných aplikací. Také

by ikona neměla být komplikovaná, tak, aby byla rozeznatelná na displejích s nízkou hustotou pixelů. Rozlišení ikony by mělo být 36 x 36 pixelů pro displeje s nízkou hustotou pixelů, 48 x 48 pixelů pro displeje se střední hustotou pixelů, 72 x 72 pixelů pro displeje s vysokou hustotou pixelů a 96 x 96 pixelů pro displeje s velmi vysokou hustotou pixelů.

Dále je potřeba vytvořit ikonu aplikace s rozlišením 512 x 512 pixelů, která je požadována pro umístění do Google Play, bude se zobrazovat uživatelům na stránce s možností stažení aplikace z Google Play.

Dále je potřeba vlastnit certifikát, kterým bude aplikace podepsána. Certifikát nemusí být vydaný certifikační autoritou, stačí si ho vygenerovat.

Pro vygenerování certifikátu lze použít nástroj keytool, který je součástí vývojového prostředí, lze zvolit ale i jednoduší způsob a využít průvodce v programu Eclipse, který provede celý proces od vygenerování certifikátu až do podepsání aplikace.

Posledním krokem je samotné sestavení aplikace a její podepsání privátním klíčem. K tomuto kroku lze využít zmíněný průvodce programu Eclipse.

Je důležité zabezpečit privátní klíč, kterým byla aplikace podepsána. Pokud by se klíč dostal do nepovolaných rukou, mohl by být zneužit k nahrazení naší aplikace aplikací osoby, která by se zmocnila našeho privátního klíče.

4.1.3 Vývojové prostředí pro Android

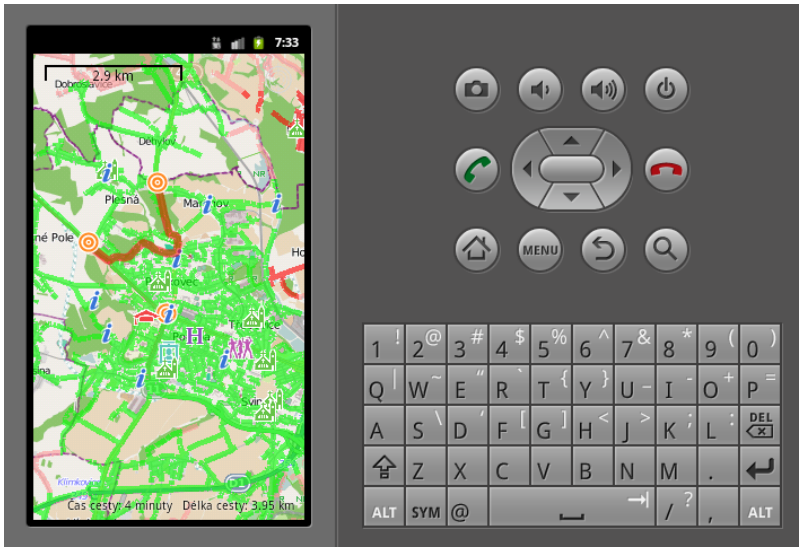
Veškeré potřebné části vývojového prostředí jsou bezplatné, jsou dostupné pro Windows, Linux i Mac OS X. [2] Pro vývoj je potřeba mít instalované:

1. JDK, nejnovější verzi lze stáhnout z [20]
2. Android Developer Tools Bundle, balík obsahující základní nástroje potřebné pro vývoj. Lze stáhnout z [10]. Obsahuje:
   - Eclipse + ADT plugin: vývojové prostředí pro Android
   - Android SDK Tools: sada nástrojů pro vývoj a ladění
   - Android Platform-tools: podpora jednotlivých Android verzí
   - Knihovny poslední Android verze
   - Obraz poslední Android verze pro emulátor

4.1.4 Android Emulátor

Pro vývoj aplikace není potřeba vlastnit fyzické zařízení s OS Android, protože vývojové prostředí zahrnuje emulátor mobilního zařízení s OS Android. Emulátor umožňuje simulovat všechny možnosti fyzického zařízení kromě uskutečnění skutečného telefonního hovoru.

Lze vytvořit emulátory s různými parametry. Lze zvolit verzi OS Android, velikost paměťové karty, rozlišení displeje, hustotu displeje a maximální velikost haldy. Také lze emulovat množství hardwarového vybavení: přídavný LCD displej, qwerty klávesnici, zvukový výstup a vstup, akcelerometr, kameru, GSM modem, senzor magnetického pole,



Obrázek 13: Vyvíjená aplikace v emulátoru

GPS modul, grafickou procesorovou jednotku a další. Na obrázku 13 je zobrazena vyvíjená aplikace v emulátoru.

Děkuji společnosti Samsung Electronics Czech and Slovak za zapůjčení tabletu Samsung Galaxy Note 10.1 (GT-N8100) za účelem testování programované aplikace.

4.1.5 Ladění aplikací

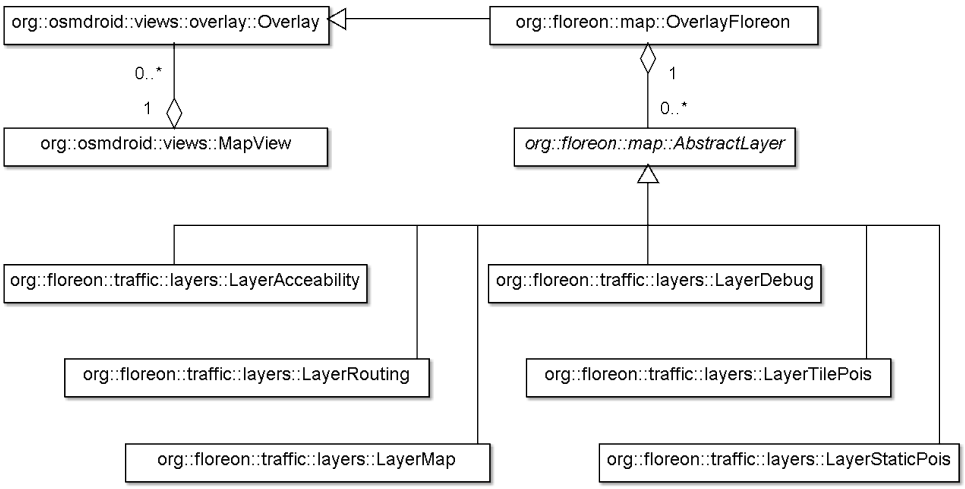
Vývojové prostředí umožňuje ladění vyvíjené aplikace. Prostředí pro ladění umožňuje krokovat program, sledovat hodnoty proměnných v průběhu krokování, nastavovat body přerušení. Lze sledovat systémové zprávy, aktuální existující vlákna procesu, stav haldy, alokaci objektů a souborový systém zařízení.

4.2 Funkce aplikace

Účel aplikace je nabídnout takovou funkčnost, která obsáhne možnosti vyvíjeného frameworku a umožní tak testovat jeho možnosti. Samotné výpočty budou probíhat na serveru, ke kterému se aplikace bude připojovat prostřednictvím webové služby popsané v kapitole 3. Samotné výsledky budou zobrazovány na mapovém podkladu OSM, který má však omezení rychlosti stahování jednotlivých dlaždic mapy.

4.2.1 Zobrazení mapových podkladů

Základem celé aplikace je obrazovka zobrazující mapu, na kterou si uživatel může volitelně nechat vykreslovat další informace (například nalezenou cestu a podobně). Jako základ pro vykreslování mapy je použita knihovna OSMDroid [19], která distribuována pod licencí Apache License 2.0.



Obrázek 14: Diagram tříd pro vykreslení dlaždic mapy

Celá mapa je rozdělena na samostatné dlaždice. Při zobrazení mapy dojde vždy první ke zjištění, které dlaždice jsou potřeba k vykreslení mapy. O samotné vygenerování potřebných dlaždic se postará instance třídy OverlayFloreon, která je vytvořena jako kontejner pro instance abstraktní třídy AbstractLayer, jejíž samotné realizace jsou konkrétní třídy, pro vykreslení specifických výsledků (například vrstva pro vykreslení cesty nebo vrstva pro vykreslení dostupnosti.) Na obrázku 14 je zachycen diagram tříd potřebných pro vykreslení dlaždic.

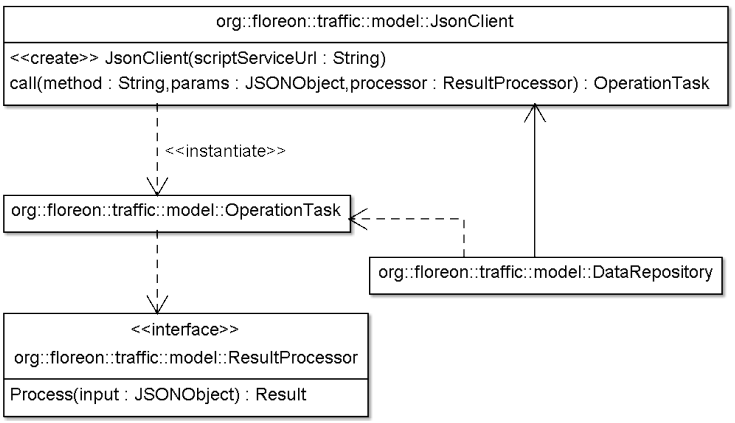
4.2.2 Připojení k webové službě

Aplikace komunikuje a webovou službou prostřednictvím zpráv ve formátu JSON. Webová služba je umístěna na adrese http://web.floreon.vsb.cz/Routing/Service.asmx. Pro komunikaci se službou byla vytvořena třída JsonClient, která umožňuje dotazy spouštět v samostatném vlákně a sledovat průběh přenosu dat, na jehož základě jsou aktualizovány prvky uživatelského rozhraní znázorňující průběh operace.

Pro zpracování výsledku se používají třídy balíčku org.json, který je součástí OS Android. Na obrázku 15 je zachycen diagram tříd pro přijení k webové službě. Interface ResultProcessor slouží ke zpracování výsledku z serveru, jeho realizace jsou součást třídy DataRepository, která slouží v aplikaci jako prezentační vrstva pro dotazy na webovou službu a jako úložiště nastavení aplikace.

4.2.3 Zpracování CSV dat

Při vykreslování dostupnosti na mapu musí být kromě samotné dostupnosti vykresleny i počáteční lokace, ze kterých se dostupnost vypočítává (tedy stanice dobrovolných nebo profesionálních hasičů). K přenosu seznamu těchto lokací se využívá format CSV, k jeho zpracování se využívá Java CSV knihovna [4] distribuovaná pod licencí LGPLv2.



Obrázek 15: Diagram tříd pro připojení k webové službě

4.2.4 Sledování funkčnosti aplikace na koncových zařízeních

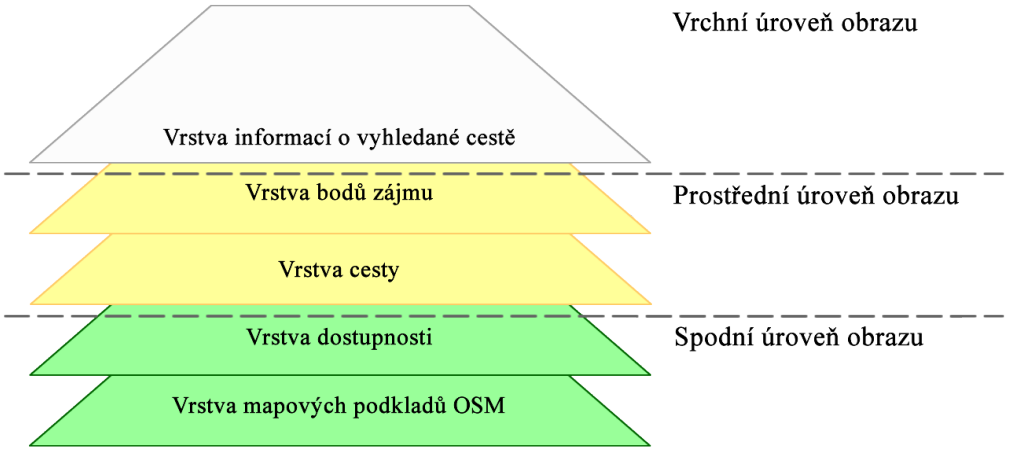
Při rozšíření aplikace mezi koncové uživatele se může stát, že dojde k neočekávané chybě aplikace, která nebyla objevena v průběhu testování. Tato chyba se může objevit například z důvodu nedůkladného otestování všech situací, rozdílným hardwarem, na kterém aplikace běží, rozdíly mezi verzemi OS Android nebo nepředvídanými událostmi za běhu OS (nedostatek paměti RAM, nedostatek paměti na paměťové kartě a další).

Proto je dobré upravit aplikaci tak, aby v případě zachycení neočekávané chyby nebo v případě pádu aplikace byl odeslán report o chybě. Takový report by měl v ideálním případě obsahovat co nejvíce informací o stavu aplikace, tedy řetězec výjimek, které chybu způsobily a technické parametry zařízení, na kterém došlo k chybě. K odeslání reportu může být využito internetové připojení zařízení.

Aplikaci využívá pro reportování chyb knihovnu ACRA [7], která je šířena pod Apache 2.0 licencí. Přidání knihovny do aplikace v nejjednodušším případě znamená přidání pouhých dvou řádků do zdrojového kódu aplikace. Knihovna umožňuje reporty zasílat více způsoby: přímo do Google dokumentů, e-mailem a na libovolný uživatelem definovaný skript.

Také umožňuje definovat způsob chování aplikace v případě nezachycené výjimky. Lze zvolit tichý mód, který zajistí odeslání reportu a ukončení aplikace. Druhý mód umožňuje v případě pádu uživateli zobrazit krátkou zprávu. Další mód zobrazuje krátkou zprávu a poté dá uživateli možnost vybrat, zdali chce odeslat report a má možnost přidat vlastní komentář. Poslední mód umožňuje navíc restartovat aplikaci.

Aplikace je nakonfigurovaná tak, aby v případě chyby byla chyba tiše odeslána. Chyby jsou odesílány na webovou službu BugSense [1], která umožňuje pohodlné zpracování odeslaných reportů. Nabízí množství grafů a statistik reportů. Všechny nové chyby (tedy ty, které se dříve nevyskytly), které jsou zachyceny, můžou být posílány na uživatelem definovaný e-mail.



Obrázek 16: Vrstvy mapy

Služba je placená, nabízí ale i bezplatnou verzi, která je omezena především maximálním počtem zachycených chyb za týden, který je u bezplatné verze 500. Pro malé projekty dostačující počet. Díky reportování chyb mohla být aplikace snadno testována v terénu.

4.3 Zpracování a vykreslení výsledků

O vykreslování mapy s výsledky se stará třída OverlayFloreon dohromady s jednotlivými vrstvami, které jsou realizacemi abstraktní třídy AbstractLayer. Výsledný obraz se skládá ze tří úrovní, spodní úroveň slouží k zobrazení dat, jejichž zpracování je náročné a vykreslují se po dlaždicích (například vypočítání dostupnosti).

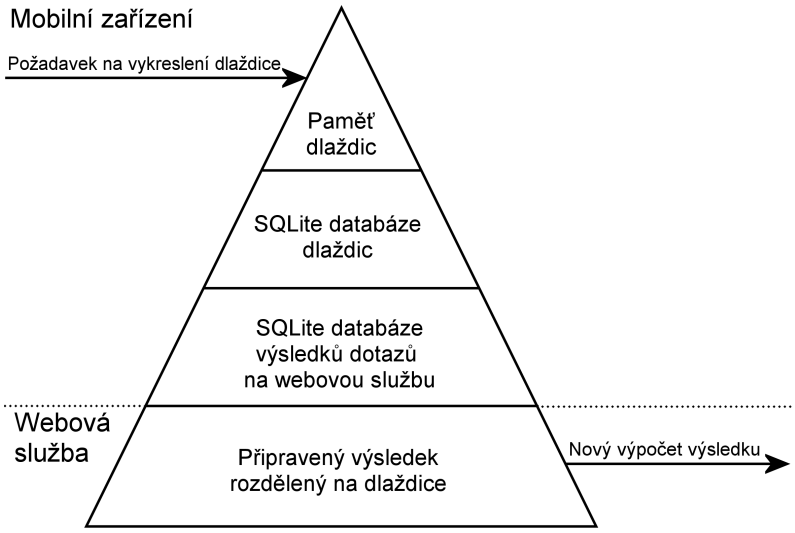
Prostřední úroveň slouží pro vykreslení dat, které nepotřebují vykreslovat po jednotlivých dlaždicích, ale lze je vykreslit pro celou obrazovku najednou (například vyhledaná cesta, body zájmu).

Vrchní úroveň slouží k vykreslení dodatečných informací, které mají statickou polohu na obrazovce a jejich poloha se nemění se změnou zobrazení mapy (například kompas, informace o délce cesty).

Jednotlivé úrovně se vykreslují pouze jednou a poté dojde k jejich překreslení v okamžiku změny zobrazení mapy nebo pokud dojde v některé z vrstev k požadavku na překreslení (například se dokončí stahování dat z webové služby a lze je vykreslit) nebo pokud OS požádá o překreslení.

Pro zajištění větší rychlosti překreslení je vytvořeno několik úrovní kešování. Část kešování se provádí ve třídě OverlayFloreon, část v samotných vrstvách vykreslující konkrétní výsledky a část přímo ve webové službě.

Na obrázku 16 jsou zobrazeny jednotlivé vrstvy mapy a jejich rozdělení na úrovně.



Obrázek 17: Využití keše při vykreslování dostupnosti

4.3.1 Spodní úroveň obrazu

Ve spodní úrovní obrazu se vykresluje dostupnost a mapové podklady OSM. Její vykreslování probíhá po jednotlivých dlaždicích mapy. Protože výpočet i vykreslení dostupnosti je časově náročná operace (vzhledem k výkonu mobilních zařízení), je vytvořen celý řetězec keší.

Pokud dojde k požadavku na vykreslení dlaždice, první v rámci třídy OverlayFloreon, dojde ke kontrole, jestli není dlaždice připravena v paměti telefonu, pokud není, pak dojde k pokusu vyhledat danou dlaždici v SQLite databázi. Pokud třída OverlayFloreon nemá k dispozici vykreslenou dlaždici, pak dojde k volání vykreslení dlaždice třídy LayerAccessability (nebo jiné realizace třídy AbstractLayer).

Zde opět dojde k pokusu vyhledat v SQLite databázi data pro zjišťovanou dostupnost (například dostupnost se stejnými parametry mohla být v minulosti již v zařízení vyhledána), pokud data dlaždice nejsou ani nyní v databázi nalezena, dojde k požadavku na webovou službu o data dané dlaždice. Webové služba používá taky keš, která je popsaná v kapitole 3.3. Obrázek 17 zachycuje všechny vrstvy keše při vykreslování dostupnosti.

V případě vykreslování mapového podkladu OSM je situace obdobná, s rozdílem kešování dlaždic mapy oproti kešování dlaždic dostupnosti. Kešovat dlaždice mapy je potřeba, protože z OSM se dlaždice stahují velice pomalu.

4.3.2 Prostřední úroveň obrazu

V prostřední úrovni obrazu se vykreslují body zájmu a nalezená cesta mezi dvěma body. Úroveň se kešuje v třídě OverlayFloreon pomocí dvou bitmap, kde jedna se vždy vykresluje na obrazovku a druhá se mezitím na pozadí připravuje, po dokončení přípravy se

úloha bitmap vymění. Vrstva bodů zájmu je navíc kešovaná v SQLite databázi, aby se nemusely opakovaně načítat jednotlivé body z webové služby.

4.3.3 Vrchní úroveň obrazu

Vrchní úroveň obrazu slouží k vykreslení informací, jejichž poloha na obrazovce nezávisí na poloze zobrazené mapy. Vykresluje informace o délce cesty a odhadované době cesty. Protože vykreslení těchto informací není časově náročná operace, není vrstva kešovaná.

4.3.4 Kešování do SQLite databáze

Aplikace využívá SQLite databázi pro kešování dat ze serveru, aby nebylo nutné o data žádat webovou službu pokaždé, když uživatel opakovaně zobrazí stejný kousek mapy, který měl zobrazený již v minulosti. Schéma databáze je zobrazeno na obrázku 18. Kešování do souborového systému je možná alternativa ke kešování do databáze, každý způsob kešování má své výhody.

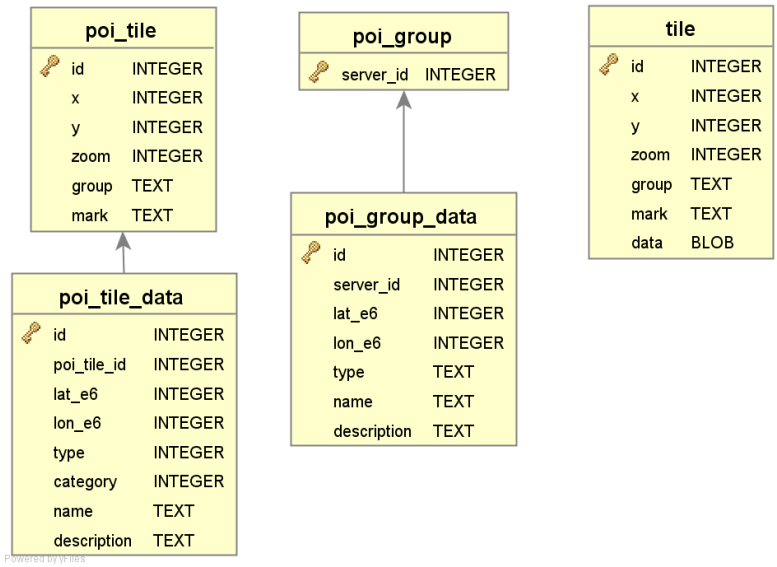
Výhodou kešování do databáze je možnost snadno přidávat různá metadata, jednoduché dotazování nad daty, snadná modifikace dat, snadno lze zjistit velikost celé keše jako velikost celého souboru databáze. Snadno lze z keše promazávat nejstarší záznamy na základě hodnot sloupce id. Výhodou kešování do souborového systému je především vyšší rychlost přístupu k datům.

Tabulka $tile$ slouží pro kešování binárních dat rozdělených podle dlaždic. Sloupce x, y a $zoom$ souží pro identifikaci dlaždice na mapě. Sloupec $group$ udává skupinu výsledků, v aplikaci se využívá jako skupina jméno třídy, která kešování provádí. Sloupec $mark$ slouží k identifikaci různých podskupin v rámci jedné skupiny. Sloupec $data$ slouží k uložení samotných binárních dat dlaždice.

Kešování do tabulky $tile$ se využívá ve vrstvě mapového podkladu, kde se ukládají všechny zobrazené dlaždice, protože samotné stahování dlaždic z OSM serveru je pomalé. Dále tabulku $tile$ využívá vrstva dopravní dostupnosti, která kešuje výsledky dostupnosti, sloupec $mark$ se využívá pro rozlišení dostupností s různým nastavením.

Tabulky poi_tile a poi_tila_data slouží pro kešování bodů zájmu rozdělených na jednotlivé dlaždice. Tabulka poi_tile má sloupce x, y, $zoom$, $group$ a $mark$, které mají stejný význam jako v tabulce $tile$. Tabulka poi_tile_data obsahuje samotné data jednotlivých bodů zájmu, tabulka obsahuje cizí klíč poi_tile_id, který ukazuje ke které dlaždici se daný bod zájmu váže. Tento druh keše se využívá ve vrstvě vykreslující body zájmu na mapu.

Tabulky poi_group a poi_group_data slouží pro kešování speciálních skupin bodů zájmu, tyto skupiny jsou identifikovány na základě sloupce $server\_id$, jehož hodnota i samotné body zájmu jsou načteny z webové služby. Tento druh keše se využívá ve vrstvě dopravní dostupnosti pro uložení skupin počátků výpočtu dostupnosti (například skupina dobrovolní hasiči a skupina profesionální hasiči).



Obrázek 18: Schéma databáze pro kešování dat

4.4 Uživatelské rozhraní

Aplikace se skládá z několika obrazovek: obrazovka mapy, obrazovka hlavního nastavení, obrazovka nastavení barev, obrazovka nastavení filtrace kategorií bodů zájmu, obrazovka nastavení vykreslované dostupnosti, obrazovka vyhledávání bodů zájmu a obrazovka informací o aplikaci.

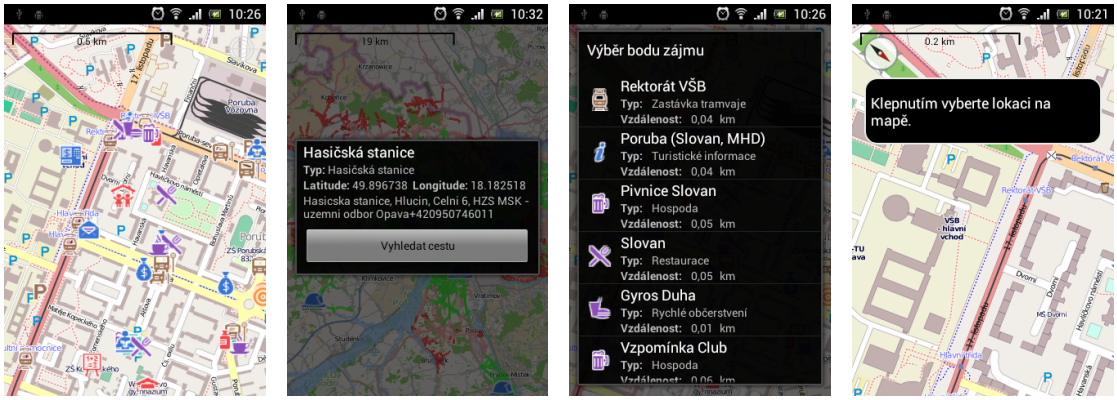
4.4.1 Obrazovka mapy

Hlavní obrazovka aplikace je obrazovka mapy, která se zobrazí uživateli po spuštění aplikace (před samotným zobrazením mapy se uživateli krátce zobrazí logo aplikace). Tato obrazovka slouží pro vykreslování výsledků, které si uživatel aktivuje, do mapy a je tedy obrazovkou, u které uživatel stráví nejvíce času a poskytuje uživateli užitečné informace.

Celou plochu obrazovky zabírá interaktivní mapa, přes kterou se zobrazují ovládací prvky přiblížení a oddálení mapy, aktivace a deaktivace GPS sledování polohy uživatele. Na mapě se zároveň vykreslují další informace, které si uživatel aktivoval (dostupnost, cesta mezi dvěma body, nalezené body zájmu).

Na mapě může uživatel používat následující gesta: dvojitým poklepáním přiblížit mapu na daný bod, dvěma prsty přibližovat a oddalovat mapu (pinch to zoom), dlouhým stiskem zobrazit kontextovou nabídku, týkající se daného místa, která umožňuje uživateli vyhledat body zájmu v daném místě nebo cestu do daného místa. Poslední gesto - klepnutí - může provádět dvě akce, záleží v jakém režimu se mapa aktuálně nachází.

Pokud je mapa v běžném režimu, klepnutím na bod zájmu na mapě se zobrazí informace o daném bodu zájmu. Klepnutím na vybarvenou oblast v rámci zobrazení výsledku



(a) Body zájmu v okolí (b) Informace o bodu (c) Volba bodu (d) Volba lokace

Obrázek 19: Obrazovka mapy

zjištění dostupnosti se zobrazí odhadovaný čas dojezdu do daného místa. Pokud je mapa v režimu výběru lokace na mapě (obrázek 19d), pak se klepnutím zvolí daná lokace (tento režim se využívá například při nastavení vyhledávané cesty).

Na obrázku 19a je zobrazena mapa s body zájmu v okolí, kategorie bodů zájmu, které se budou vykreslovat lze nastavit v nastavení aplikace. Klepnutím na libovolný zobrazený bod zájmu se zobrazí dostupné informace o daném bodě (obrázek 19b). Jeli v daném místě mapy více bodů zájmu blízko u sebe, pak nelze přesně určit, na který bod zájmu uživatel klepl a bude mu zobrazena nabídka s volbou bodu (obrázek 19c).

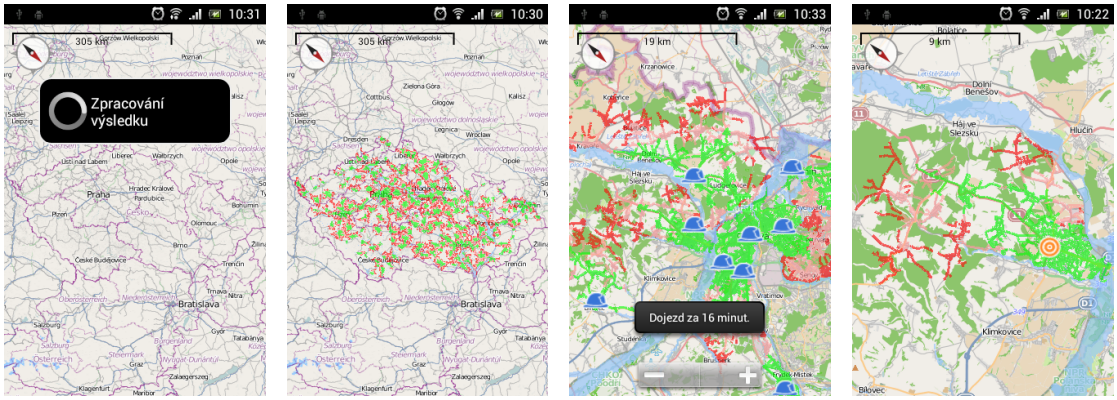
Zpracování výsledku dostupnosti je časově dlouhá operace, proto se uživateli v průběhu zobrazuje informace, že probíhá zpracování (obrázek 20a). Po zpracování výsledku se uživateli zaměří mapa tak, aby viděl celou oblast relevantní k vypočítané dostupnosti (obrázek 20b).

Při dostatečném přiblížení mapy se uživateli zobrazí i počáteční místa, které jsou brány jako počáteční body pro výpočet dostupnosti - na obrázku 20c jsou zobrazeny hasičské stanice (klepnutím na hasičskou stanici lze zjistit přesnější informace o stanici - obrázek 19b) a na obrázku 20d je zobrazen uživatelem definovaný počátek.

Na obrázku 20c i 20d je zobrazena dostupnost s nastavenými nedostupnými oblastmi při stoleté vodě. Klepnutím na mapu v místě vykreslené dostupnosti se zobrazí informace o době dojezdu do daného místa, popřípadě informace, že se jedná o zatopenou oblast (obrázek 19b).

Při vyhledávání cesty mezi dvěma body buď uživatel vybere daný bod zájmu a v jeho detailu vybere možnost vyhledat cestu (obrázek 19b) nebo dlouhým stisknutím na mapě otevře kontextovou nabídku a vybere možnost najít cestu.

Poté se mu otevře okno pro zadání, odkud se cesta bude hledat (obrázek 21a), v tomto okně může počátek zadat buď jako souřadnice, vybrat klepnutím na mapě, načíst aktuální polohu z GPS nebo vyhledat bod zájmu v okolí nebo pomocí fulltextu. Po vypočítání cesty se uživateli zaměří mapa na nalezenou cestu (obrázek 21b).



(a) Zpracování (b) Celá dostupnost (c) Detail dostupnosti (d) Dostupnost z bodu

Obrázek 20: Zobrazení dostupnosti na mapě

4.4.2 Obrazovka vyhledání bodu zájmu

Pokud zvolí uživatel funkci hledání bodů zájmu, zobrazí se mu obrazovka rozdělená na tři záložky, v první záložce může hledat body zájmu v okolí s možností filtrovat podle daných kategorií (obrázek 21c). Na druhé záložce může uživatel vyhledat body zájmu pomocí fulltextu (obrázek 21d) a na poslední záložce může najít adresy, které se nacházejí v jeho okolí.

4.4.3 Ostatní obrazovky

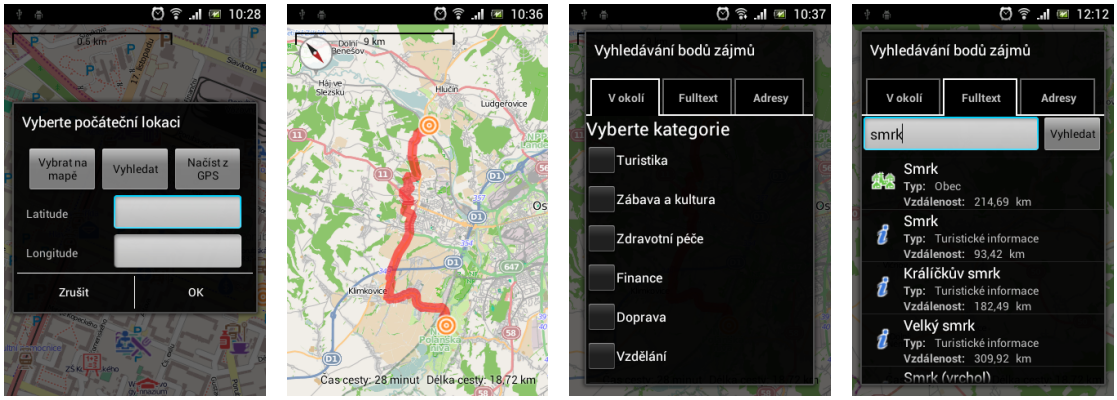
Další obrazovka aplikace je obrazovka základního nastavení, ve které může uživatel nastavit maximální rychlost vozidla, jestli chce při vyhledávání používat i placené úseky cest a jestli se jedná o nákladní vozidlo. Také může nastavit maximální velikost keše. Další obrazovka je obrazovka nastavení barev při vykreslování dostupnosti (každý čas dostupnosti má svoji barvu).

4.5 Experimentální otestování fulltextového vyhledávání

Bude provedena řada experimentů, jejichž cílem bude ověřit správnost řazení nalezených bodů zájmu při fulltextovém vyhledávání. Správným řazením je myšleno seřazení takovým způsobem, aby na vyšších pozicích nalezených bodů zájmu byly body, které se více podobají zadanému vstupu a jsou pro uživatele důležitější. Pro otestování bude vytvořeno několik fulltextových dotazů a bude určeno jestli nalezené body zájmu odpovídají dotazu.

Prvním dotazem je otestován vliv počtu slov v jménech bodů zájmu. Hledaný výraz je „Ostrava“. Výsledek je: Ostrava; Moravská Ostrava; Nordica Ostrava; ... Lze vidět, že bod zájmu který odpovídal svojí délkou byl řazen výše.

Další dotaz testuje možnost zadat dotaz s diakritikou i bez diakritiky a libovolně malými i velkými písmeny. Hledaný dotaz je „hrabůvka“ a „HRABUVKA“, pro oba dotazy



(a) Počátek cesty (b) Nalezená cesta (c) Nejbližší body (d) Fulltext

Obrázek 21: Zobrazení cesty na mapě a hledání bodů zájmu

je výsledek: Hrabůvka (část obce); Hrabůvka (obec); Poliknika Hrabůvka; ... Výsledek odpovídá očekávání.

Další dotaz otestuje chybějící a přebytečné písmeno. Uživatel bude chtít například hledat „Ostrava“, ale zadá „otravaa“. Výsledek je: Odrava; Ostrava; Otava; Moravská Ostrava; ... Levenshteinova vzdálenost mezi zadaným textem a jménem nalezeného bodu zájmu pro první tři záznamy je tři, proto mají všechny tři stejnou váhu a proto se záznam „Ostrava“ může nacházet na libovolné pozici z prvních tří. Čtvrtý záznam obsahuje dvě slova (zadaný text pouze jedno), projevil se vliv počtu slov, testovaný v prvním testovacím dotazu. Výsledek odpovídá očekávání.

Další dotaz otestuje překlepy. Uživatel bude opět chtít najít město „Ostrava“, ale zadá „oxtxaxa“. Výsledek: Ostrava; Metaxa; Otava; Ostrata; Moravská Ostrava; ... První tři záznamy mají Levenshteinovu vzdálenost tři, třetí záznam čtyři a u dalšího záznamu se projevil počet slov ve jméně bodu zájmu.

Dotaz „Ostrava Moravská“ otestuje hledání fráze obsahující více slov. Výsledek: Moravská Ostrava; Moravská Ostrava 3130; Moravská Ostrava 1278; Moravská Ostrava 3109; ÚMOb Moravská Ostrava a Přívoz; ... Výsledek obsahuje na prvním místě bod názvu, který odpovídá zadání a na dalších pozicích jsou pak záznamy mající větší počet slov.

Posledním dotaz „mAArykkavo nmestío“ je kombinací předchozích testů, jeho výsledek je podle očekávání: Masarykovo náměstí.

Na základě experimentálních testů lze předpokládat, že fulltextové vyhledávání nabízí relevantní výsledky, v rámci dat dostupných z OSM.

5 Závěr

Byla implementována datová struktura trie. Trii je možné ukládat a načítat ze souboru na disku. Bylo implementováno přesné a nepřesné vyhledávání v trii, které bylo použito pro realizaci fulltextového vyhledávání bodů zájmu.

Byly hledány optimalizace fulltextového vyhledávání. Optimalizace volbou různých způsobů ukládání uzlů v trii neměla velký vliv na rychlost vyhledávání, ale snížila paměťovou složitost trie. Optimalizace snížením počtu alokací proměnných potřebných pro výpočet Levenshteinovy vzdálenosti zvýšila rychlost vyhledávání pouze mírně, ale optimalizace změnou způsobu nepřesného vyhledávání v trii, tak, aby byly uzly procházeny v pořadí od nejnižší Levenshteinovy vzdálenosti, způsobila zásadní snížení doby vyhledávání.

Byla implementována datová struktura k-d strom. K-d strom je možné ukládat a načítat ze souboru na disku. Bylo implementováno vyhledávání bodů zájmu nejbližších k zadaným souřadnicím a bodů zájmu v zadané oblasti mapy. Vyhledávání umožňuje filtrovat vyhledávané body zájmu podle kategorií (například kategorie vzdělání, zábava a další). Byl proveden test rychlosti vyhledávání, doby vyhledávání v testu se pohybovaly ve většině případů pod 1 ms, což lze považovat za dobrý výsledek.

Byla implementována a nasazena webová služba, která využívá implementované vyhledávání bodů zájmu a framework pro směrovaní vyvíjený VŠB-TU Ostrava. Webová služba umožňuje vyhledávat body zájmu, vyhledávat nejkratší cestu mezi dvěma body a vypočítat dostupnost z jednoho nebo více počátečních míst.

Nakonec byla vyvinuta aplikace pro operační systém Android, která využívá vytvořenou webovou službu. Aplikace demonstruje některé možnosti vyvíjeného frameworku.

Další vývoj projektu by mohl směřovat například v dalších optimalizacích algoritmu fulltextového vyhledávání, protože jeho rychlost je aktuálně nejnižší ze všech implementovaných algoritmů. Optimalizace by mohla spočívat například ve využití keše, kdy by se kešovaly vyhledávané dotazy a jejich výsledky. Při opakovaném položení dotazu by se výsledek načetl rychle z keše, místo pomalého vyhledávání v trii.

Dále by mohl algoritmus fulltextového vyhledávání být rozšířen o možnost vyhledávat na základě uživatelovy lokace, kdy by výsledky, které jsou k uživatelově lokaci blíže, byly řazeny výše, takže například pokud ve světě existuje více měst se stejným jménem, pak by uživatel viděl na prvním místě to město, které je mu blíže.

Vyhledávání nejbližších bodů zájmu by mohlo být rozšířeno o možnost řadit výsledky podle cestovní vzdálenosti mezi středem vyhledávání a jednotlivými nalezenými body místo vzdálenosti vzdušné.

Samotná aplikace pro operační systém Android by mohla být v budoucnu použita jako základ navigace, včetně hlasových pokynů k jízdě nebo aplikace pro geocaching a jiné. Mohla by být přidána funkčnost pro uživatelem definované omezení při výpočtu dostupnosti, například uživatel by si mohl zadat, že daná silnice je neprůjezdná a nechal by si vypočítat, kam všude dojede v zadaném časovém intervalu ze zadaného bodu.

6 Reference

[1] BUGSENSE, Inc. *BugSense* [online]. 2013 [cit. 2013-04-18]. Dostupné z: https://www.bugsense.com/

[2] BURNETTE, Ed. *Hello, android: introducing Goggle's mobile development platform*. 3rd ed. Raleigh, N.C.: Pragmatic Programmers, 2010, xviii, 293 p. ISBN 978-193-4356-562.

[3] DASGUPTA, Sanjoy; SINHA, Kaushik. Randomized partition trees for exact nearest neighbor search. *arXiv:1302.1948*, 2013.

[4] DUNWIDDIE, Bruce. *Java CSV Library* [online]. 2012 [cit. 2013-04-03]. Dostupné z: Http://sourceforge.net/projects/javacsv/

[5] DVORSKÝ, Jiří. *Algoritmy I.* Ostrava, 2007. VŠB - Technická univerzita Ostrava. Dostupné z: http://www.cs.vsb.cz/dvorsky/

[6] GAEDE, Volker; GÜNTHER, Oliver. *Multidimensional Access Methods*. Berlín, 1998. Humboldt-Universität zu Berlin.

[7] GAUDIN, Kevin. *ACRA - Know your BUGS* [online]. 2013 [cit. 2013-04-18]. Dostupné z: http://acra.ch/

[8] Android, the world's most popular mobile platform. GOOGLE INC. *Android Developers* [online]. 2013 [cit. 2013-03-03]. Dostupné z: http://developer.android.com/about/index.html

[9] Dashboards. GOOGLE INC. *Android Developers* [online]. 2013 [cit. 2013-03-03]. Dostupné z: http://developer.android.com/about/dashboards/index.html

[10] Get the Android SDK. GOOGLE INC. *Android Developers* [online]. 2013 [cit. 2013-01-15]. Dostupné z: http://developer.android.com/sdk/index.html

[11] HSU, Bo-June Paul; OTTAVIANO, Giuseppe. *Space-Efficient Data Structures for Top-k Completion*. Pisa, 2013. Dostupné z: http://www.di.unipi.it/~ottavian/files/topk_completion_www13.pdf. To appear in Proceedings of the 22st World Wide Web Conference. Università di Pisa.

[12] KNUTH, Donald Ervin. *The art of computer programming*. 3rd ed. Upper Saddle River: Addison-Wesley, 1998, xiii, 780 s. ISBN 02-018-9685-0.

[13] KRÁTKÝ, Michal; SKOPAL, Tomáš; SNÁŠEL, Václav. *Vícerozměrný přístup pro netriviální vyhledávání termů*. Ostrava, 2003. VŠB – Technická univerzita Ostrava.

[14] KURTZ, Stefan. *Reducing the Space Requirement of Suffix Trees* [online]. Postfach, 1999 [cit. 2013-04-15]. Dostupné z: http://www.cs.cmu.edu/afs/cs/project/aladdin/wwwlocal/hash/Ku99.pdf. Software-Practice and Experience. Universitat Bielefeld.

[15] KÜHN, Stefan. *Tissot's Indicatrixs by Mercator-Projection* [online]. 2004. Dostupné z: https://commons.wikimedia.org/wiki/File:Tissot_mercator.png

[16] Bing Maps Tile System. MICROSOFT. *Microsoft Developer Network* [online]. 2009 [cit. 2013-04-02]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb259689.aspx

[17] MOORE, Andrew. *A tutorial on kd-trees*. Cambridge, 1991. Dostupné z: http://www.cs.cmu.edu/*sim*awm/papers.html. Extract from PhD Thesis. University of Cambridge.

[18] Mapnik. OPENSTREETMAP. *OpenStreetMap Wiki* [online]. 2013 [cit. 2013-04-02]. Dostupné z: http://wiki.openstreetmap.org/wiki/Mapnik

[19] OSMDROID. *OSMDroid OpenStreetMap - Tools for Android* [online]. 2013 [cit. 2013-04-03]. Dostupné z: code.google.com/p/osmdroid/

[20] Java SE Downloads. ORACLE CORPORATION. *Oracle* [online]. 2013 [cit. 2013-01-15]. Dostupné z: http://www.oracle.com/technetwork/java/javase/downloads/index.html

[21] A Brief History of Android. SPICE STELLAR. *Infographics & Data Visualization* [online]. 2013 [cit. 2013-03-03]. Dostupné z: http://visual.ly/brief-history-android

[22] STATCOUNTER. *StatCounter Global Stats* [online]. 2013 [cit. 2013-03-03]. Dostupné z: http://gs.statcounter.com

[23] Web Services Architecture. WORLD WIDE WEB CONSORTIUM. *World Wide Web Consortium (W3C)* [online]. 2004 [cit. 2013-04-02]. Dostupné z: http://www.w3.org/TR/ws-arch/

[24] Points of Interest Core. WORLD WIDE WEB CONSORTIUM. *World Wide Web Consortium (W3C)* [online]. 2011 [cit. 2013-03-11]. Dostupné z: http://www.w3.org/TR/poi-core/